ioProgrammo Anno XI - N°5 (114)

AGGIORNA IL BLOG CON IL CELLULARE • FATTI IL GADGET PER WINDOWS VISTA • E-COMMERCE CON DRAG & DROP • INTRODUZIONE A JYTHON

EDIZIONI MASTER

PROGRAMMAZIONE AD OGGETTI

LE TECNICHE PER PROGETTARE LE TUE APPLICAZIONI IN MODO CHE SIANO STABILI E FACILMENTE MANUTENIBILI!

VERSIONE PLUS RIVISTA+LIBRO+CD €9,90 | VERSIONE STANDARD RIVISTA+CD €6,90

ioPROGRAMMO

PER ESPERTI E PRINCIPIANTI

Poste Italiane S.p.A Spedizione in A.P. • D.L. 353/2003 (conv.in L.27/02/2004 n.46) art.1 comma 2 DCB ROMA **Periodicità mensile • MAGGIO 2007 • ANNO XI, N.5 (114)**

Aggiorna il BLOG con il CELLULARE

Scrivi un'applicazione J2ME ed invia le news sul web in ogni momento e da qualunque parte del mondo

USERNAME E PASSWORD salvali in un "portachiavi" per non doverli riscrivere ogni volta

SCRITTURA E UPLOAD memorizza i post localmente per inviarli tutti insieme in un colpo solo

DOWNLOAD E LETTURA sincronizza i dati locali con quelli remoti ogni volta che ti connetti

GESTIONE DEL SERVER modifica il tuo sito PHP per gestire le richieste provenienti dal telefonino

FATTI IL GADGET PER WINDOWS VISTA

Arrivano le "miniapplicazioni" residenti sul desktop. Ecco come crearne una che ti mostra costantemente lo spazio disponibile e occupato sul tuo Hard Disk

WEB

E-COMMERCE CON DRAG & DROP

Usa Ajax e metti un prodotto nel carrello semplicemente trascinandolo sulla sua icona

SISTEMA

DALLE INTERFACCE AI DATABASE CON JYTHON

Arriva il linguaggio che fonde insieme Python e Java, ecco come funziona

AUTOMI CELLULARI E GIOCO DELLA VITA

algoritmi per simulare la diffusione di un fenomeno

WEB

MA QUANTE VISITE HA IL MIO SITO?

Scoprilo analizzando direttamente i LOG di IIS ed elaborando grafici statistici

SITI MULTILINGUA CON STRUTS 2

Localizza i tuoi contenuti con il framework Java per il Web più usato al mondo

USA JAVASCRIPT DA OGNI DOMINIO

Ecco come utilizzare anche gli script presenti su un sito esterno al tuo

PATTERN

L'ADAPTER METTE TUTTI D'ACCORDO

Prendi pezzi di codice da Internet e falli comunicare insieme con questo pattern

C++

GESTIRE LE STRINGHE IN MODO SICURO

Metti al riparo i tuoi programmi dai possibili attacchi degli hacker

msdn WEBCAST **3 NUOVI VIDEO** WINDOWS PRESENTATION FOUNDATION - WPF

MOBILE

USA AJAX NEI DISPOSITIVI MOBILI

Progetta siti pronti per il Web 2.0 e compatibili con cellulari e smartphone

UN SMS SERVER CON WINDOWS MOBILE

Il tuo Pocket PC reagisce ai messaggi in arrivo e si comporta come vuoi tu

EDIZIONI MASTER www.edmaster.it

9 771128 594009 70114

30-03-2007 17:01:23

Anno XI - N.ro 05 (114) - Maggio 2007 - Periodicità Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale: Massimo Sesti
Direttore Responsabile: Massimo Sesti
Responsabile Editoriale: Gianmarco Bruni
Vice Publisher: Paolo Soldan
Redazione: Fabio Farnesi
Collaboratori: R. Allegra, L. Buono, D. De Michelis, F. Grimaldi, E. Viale, F. Cozzolino, I. Venuti, O. Peli
Segreteria di Redazione: Rossana Scarcelli

Realizzazione grafica: Cromatika S.r.l.
Art Director: Paolo Cristiano
Responsabile grafico di progetto: Salvatore Vuono
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri
Impaginazione elettronica: Francesco Cospite, Lisa Orrico, Nuccia Marra, Luigi Ferraro

Realizzazione Multimediale: SET S.r.l.
Realizzazione CD-Rom: Paolo Iacona

Pubblicità: Master Advertising s.r.l.
Via C. Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite: Daisy Zonato

Editore: Edizioni Master S.p.a.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Presidente e Amministratore Delegato: Massimo Sesti
Direttore Generale: Massimo Rizzo

ABBONAMENTO E ARRETRATI

ITALIA: Abbonamento Annuale: ioProgrammo (11 numeri) €59,90 sconto 21% sul prezzo di copertina di €75,90 · ioProgrammo con Libro (11 numeri) €75,90 sconto 30% sul prezzo di copertina di €108,90 Offerte valide fino al 30/06/07.
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via C. Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito Visa, Cartasì, o Eurocard/Mastercard (inviando la Vs. autorizzazione, il numero di carta di credito, la data di scadenza, l'intestatario della carta e il codice CVV2, cioè le ultime 3 cifre del codice numerico riportato sul retro della carta).
- bonifico bancario intestato a Edizioni Master S.p.A. c/o BCC MEDIOCRATI S.C.AR.L. c/c 0 000 000 12000 ABI 07062 CAB 80880 CIN P (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.

Sostituzioni: qualora nei prodotti fossero rinvenuti difetti o imperfezioni che ne limitassero la fruizione da parte dell'utente, è prevista la sostituzione gratuita, previo invio del materiale difettato.
La sostituzione sarà effettuata se il problema sarà riscontrato e segnalato entro e non oltre 10 giorni dalla data effettiva di acquisto in edicola e nei punti vendita autorizzati, facendo fede il timbro postale di restituzione del materiale.
Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master - Servizio Clienti - Via C. Correnti, 1 - 20123 Milano

Servizio Abbonati:
☎ tel. 02 831212
@ e-mail: servizioabbonati@edmaster.it

Assistenza tecnica: ioprogrammo@edmaster.it
Stampa: Arti Grafiche Boccia S.p.a. Via Tiberio Felice, 7 Salerno
Stampa CD-Rom: Neotek S.r.l. - C.da Imperatore - Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Aprile 2007

1 Anno di Computer Bild 2006, 1 Anno di Io Programmo in DVD 2006, 1 Anno di Linux Magazine in DVD 2006, 1 Anno di Office Magazine 2006, 1 Anno di Win Magazine in DVD 2006, Audio/Video/Foto Bild Italia, A-Team, Calcio & Scommesse, Colombo, Computer Bild Italia, Computer Games Gold, Digital Japan Magazine, Digital Music, Distretto di Polizia in DVD, DVD Magazine, DVD Magazine Films, Family DVD Games, Filmteca in DVD, Go!OnLine Internet Magazine, Home Entertainment, Horror Mania, I Film di DVD Magazine, I DVD di Win Magazine, I DVD de La Mia Barca, I Film di Idea Web, I Filmissimi in DVD, I Film di DVD Magazine, I Grandi Giochi per Pc, I Libri di Quale Computer, I Mitici all'Italiana, Idea Web, Idea Web Film, InDVD, IoProgrammo, I Tecnoplus di Win Magazine, Japan Cartoon, La mia Barca, La mia Videoteca, Le Femme Fatale del Cinema, Le Grandi Guide di Io Programmo, Linux Magazine, Magnum PI, Miami Vice in DVD, Nightmare, Office Magazine, Play Generation, Play Generation Games, Popeye, PC Junior, Quale Computer, Softline Software World, Sport Life, Supercar in DVD, Star in DVD, Video Film Collection, Win Junior, Win Magazine Giochi, Win Magazine, Le Collection.

Questo mese su ioProgrammo

▼ "SLOW" PROGRAMMING

La novità più importante del mese è quella relativa all'annuncio dei piani di sviluppo per Symbian OS. La nuova versione del sistema operativo più diffuso per i sistemi embedded è prevista per il 2009. La direzione intrapresa è quella della diminuzione delle richieste hardware, un miglior supporto al VOIP, l'inserimento di funzioni relative allo streaming multimediale anche dalla rete, in altre parole TV Over IP.
Non possiamo non fare qualche considerazione rispetto a questo annuncio, non tanto nel merito della notizia, quanto sulle indicazioni generali rispetto alla programmazione che scaturiscono dai suoi contenuti. Prima di tutto: i tempi. Il piano di sviluppo viene annunciato a 2 anni di distanza dalla data di rilascio. Due anni rappresentano un'eternità in materia di programmazione e di informatica in generale o almeno la rappresentano rispetto alle comuni abitudini dei nostri clienti. Chiedere due anni di sviluppo per un progetto equivarrebbe per noi a perdere la commessa, mentre rappresentano un tempo "normale" per una casa come Symbian. L'attenzione alla diminuzione delle richieste hardware. Anche in questo caso per noi non rappresenterebbe un problema. E infine la lungimiranza verso tecnologie che attualmente sono già diffuse ma che si prevede saranno di uso comune proprio in un tempo informatico abbastanza lungo. Tutto questo ci deve far riflettere su quanto la programmazione abbia necessità di tornare ad una dimensione più umana, dove il tempo di sviluppo è commisurato alle necessità del cliente ma anche alle reali difficoltà del progetto. Non sempre aggiungere una nuova risorsa significa diminuire il tempo di sviluppo. In conclusione è importante nel corso dei nostri progetti aumentare lo spazio riferito all'analisi, ai requisiti e alla gestione delle risorse, solo così riusciremo a tirar fuori progetti efficienti.

All'inizio di ogni articolo, troverete un simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre \soft\codice\ e \soft\tools\) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.

AGGIORNA IL BLOG CON IL CELLULARE

Scrivi un'applicazione J2ME ed invia le news sul Web in ogni momento e da qualunque parte del mondo

Username e Password
salvali in un "portachiavi" per non doverli riscrivere ogni volta

Scrittura e Upload
memorizza i post localmente per inviarli tutti insieme in un colpo solo

Download e Lettura
sincronizza i dati locali con quelli remoti ogni volta che ti connetti

Gestione del server
modifica il tuo sito PHP per gestire le richieste provenienti dal telefonino

pag. 16

FATTI IL GADGET PER WINDOWS VISTA

Arrivano le "miniapplicazioni" residenti sul desktop. Ecco come crearne una che ti mostra costantemente lo spazio disponibile e occupato sul tuo Hard Disk

IOPROGRAMMO WEB

ANTEPRIMA

MOBILE

SISTEMA

PATTERN

SOLUZIONI

RUBRICHE

QUALCHE CONSIGLIO UTILE

I nostri articoli si sforzano di essere comprensibili a tutti coloro che ci seguono. Nel caso in cui abbiate difficoltà nel comprendere esattamente il senso di una spiegazione tecnica, è utile aprire il codice allegato all'articolo e seguire passo passo quanto viene spiegato tenendo d'occhio l'intero progetto. Spesso per questioni di spazio non possiamo inserire il codice nella sua interezza nel corpo dell'articolo. Ci limitiamo a inserire le parti necessarie alla stretta comprensione della tecnica.

http://forum.ioprogrammo.it

I contenuti del libro

Lavorare con Java 6

Java festeggia più di 10 anni di vita. Nel tempo è diventato un linguaggio dalla complessità enorme, completato da un ecosistema di librerie veramente imponente.
Al contempo la sua diffusione ha raggiunto vette decisamente importanti. Va da se che ogni nuova versione è accompagnata da problemi di retrocompatibilità ma anche foriera di importanti novità, che la rendono aderente agli sviluppi che i sistemi operativi e più in generale l'informatica attraversano nel corso del tempo.
In questo libro Ivan Venuti, non solo mostra un approccio decisamente pratico alla risoluzione di problemi di programmazione giornalieri, ma illustra come le nuove funzionalità di Java 6 possano decisamente innalzare la produttività di ogni sviluppatore. Si tratta di un HandBook veloce, pratico, essenziale che consente un viaggio pratico e divertente nell'universo Java

PER SFRUTTARE A FONDO TUTTE LE CARATTERISTICHE DELLA NUOVA VERSIONE DEL LINGUAGGIO DI SUN

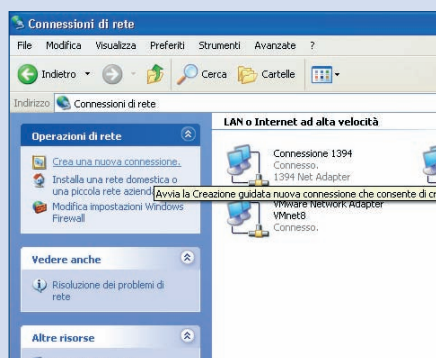
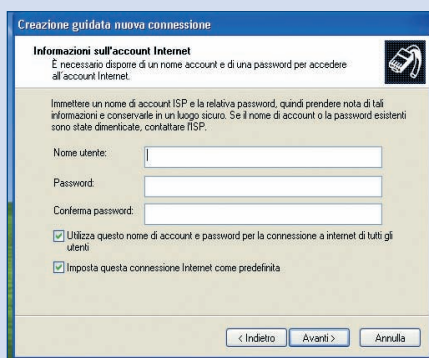
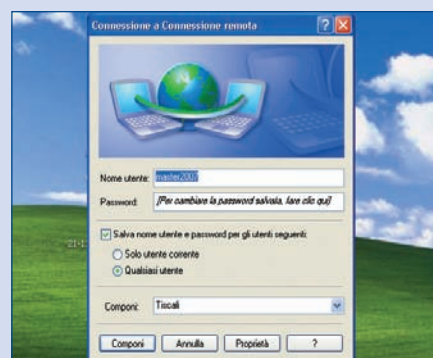
- **Installazione e novità**
- **applicazioni Desktop & Grafica**
- **Database & Scripting**

Online con Tiscali Easy

Numero unico per tutta l'Italia e nessuna registrazione. Il modo più semplice e veloce per entrare in Internet risparmiando

Sei spesso lontano da casa e hai necessità di collegarti a Internet ovunque ti trovi? Desideri salvaguardare la tua privacy navigando in modo anonimo? Non hai voglia di perdere tempo in lunghe registrazioni per creare un nuovo account? Niente paura, Tiscali ha pensato anche a te! Con Tiscali Easy arriva un sistema tutto nuovo di collegarsi a Internet. Non sarà più necessario creare un nuovo account e fornire i propri dati personali, potrai navigare collegandoti con un unico numero di telefono (7023456789) da tutta Italia e soprattutto utilizzando i dati di accesso forniti direttamente da Tiscali (UserID e Password presenti sulla scheda), quindi non collegabili in alcun modo a te. Insomma, con Tiscali Easy, otterrai in un colpo solo riservatezza dei dati, risparmio del tempo necessario alla creazione di un abbonamento e tariffe vantaggiose. I costi di connessione sono simili a quelli di un classico abbonamento gratuito: 1,90 cent. per i primi 10 minuti e 1,72 cent. per quelli successivi. Per risparmiare ulteriormente è possibile connettersi a Internet durante le ore serali o nei giorni festivi al costo di 1,09 cent. per i primi dieci minuti e 0,98 cent. per quelli successivi. L'unico requisito richiesto per poter eseguire la connessione è un modem correttamente installato. Poiché si tratta di una normale connessione analogica è possibile utilizzare i tool di accesso remoto integrati in Windows

IN RETE CON TISCALI

Nessuna registrazione basta creare una nuova connessione e inserire i dati di accesso forniti da Tiscali

1 NUOVA CONNESSIONE
Portiamoci nel pannello di controllo, e selezioniamo l'icona connessioni di rete. In alto a sinistra selezioniamo "nuova connessione" e prepariamoci a seguire il wizard che comparirà

2 DATI DI ACCESSO
Seguiamo il Wizard fino a quando non ci vengono chiesti i dati per l'autenticazione. E' sufficiente inserire quelli presenti nella card allegata a questo numero di ioProgrammo: master2007/tiscali

3 ACCESSO A INTERNET
Connettersi è semplicissimo, troveremo una nuova icona all'interno del pannello di controllo alla voce connessioni. Bastano due click ed il gioco è fatto sarete connessi ovunque vi troviate

RIVISTA + CD-ROM in edicola

VIRTUAL BOX 1.3.8

Il virtualizzatore Free

La virtualizzazione è l'ultima frontiera. La nutrita schiera di sistemi operativi con le relative versioni, rende particolarmente complesso lo sviluppare programmi in grado di funzionare sui vari sistemi senza problemi. Per questo ogni programmatore tende a testare le proprie applicazioni su più sistemi. E' ovviamente quasi impossibile disporre di tante macchine per quanti sono i sistemi operativi così ci vengono in aiuto i software di virtualizzazione che emulano il comportamento di una macchina via software. VirtualBox è uno di questi. E' recente, è molto leggero ed è free.

Directory:/VirtualBox_1.3.8_Win_x86.msi

Prodotti del mese

Crystal Space 1.0

Il re c'è ancora

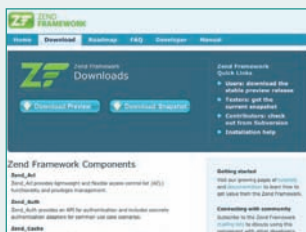
È stato per lungo tempo il dominatore delle scene per quanto riguarda gli Engine 3D. Lo è stato almeno fino all'avvento di Irrlicht e del più recente XNA. Questo non vuol dire che non rappresenti ancora lo stato dell'arte nel campo della programmazione dei videogiochi. Ad oggi si tratta di uno degli engine 3D che conta il maggior numero di installazioni e giochi sviluppati. In questo numero vi presentiamo la release 1.0 di cui purtroppo sono disponibili solo i sorgenti. Tuttavia la compilazione non è un'operazione complicata e vi mette in grado di conoscere ancora meglio il prodotto su cui state lavorando. Il framework è piuttosto veloce e si possono ottenere effetti straordinari con uno sforzo minimo. Non per niente Crystal Space è stato per lungo tempo il più diffuso.

[pag.106]

Zend Framework 0.9

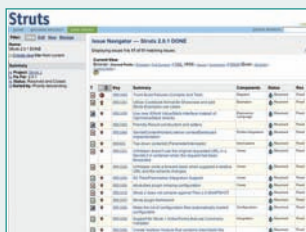
L'SDK evoluto per PHP

Chi sviluppa in PHP è abituato a sviluppare in maniera autonoma il proprio framework, partendo dalle proprie esperienze e necessità. Questo ha però dato origine ad un ecosistema di applicazioni spesso difficilmente manutenibili. Zend ha sviluppato un proprio Framework completo che dispone di una serie di meccanismi che velocizzano la risoluzione dei problemi più frequenti. Si va dall'implementazione del pattern MVC alla creazione dei Web Services, alla gestione delle stampe in PDF. Si tratta di un framework piuttosto affidabile essendo sviluppato da Zend che è anche la software house promotrice dello sviluppo di PHP. Si tratta di un modo innovativo di guardare a PHP, che fino ad ora non prevedeva un framework unitario

[pag.110]

Struts 2.0.1

Il framework MVC per Java

Il pattern MVC è probabilmente il più usato da tutti gli sviluppatori in ogni linguaggio.Questo framework è il leader per quanto riguarda lo sviluppo di applicazioni Java secondo il pattern MVC. Molto comodo, stabile e affidabile, rappresenta la base di partenza per applicazioni che devono essere facilmente manutenibili.Uno standard da usare se progettate applicazioni di dimensioni generose, ma anche quando si sviluppano web application professionali. Non si tratta sicuramente di un framework facile da usare, non per niente in queste pagine vi dedichiamo spesso approfondimenti, ma è anche vero che una volta appresa la logica di funzionamento, la tipologia di applicazioni risultante appare realmente solida. La difficoltà sta tutta nella curva di apprendimento, una volta superato il gradino iniziale tutto risulta sufficientemente omogeneo

[pag.109]

Irrlicht 1.2

Accendi il migliore motore 3D Open Source!

Irrlicht è un motore per la grafica tridimensionale, scritto in C++ e utilizzabile sia con questo linguaggio, sia con la tecnologia .NET.Presenta le principali caratteristiche di un motore professionale e vanta una notevole comunità di sviluppatori, con diversi progetti in attivo. Irrlicht ha tra i suoi pregi anche quello di potere utilizzare sia DirectX che OpenGL per cui diventa particolarmente adatto anche per lo sviluppo di giochi multipiattaforma. Nel tempo Irrlicht ha assunto dimensioni particolamente rilevanti fino a diventare quasi uno standard per chi sviluppa VideoGames. I suoi connotati essenziali sono la curva di apprendimento molto rapida e la completezza delle funzionalità esposte, che consentono una flessibilità rara in questo mercato

[pag.106]

GLI ALLEGATI DI IOPROGRAMMO

Questo mese tre Webcast per comprendere le novità incluse nell'ambiente grafico di Windows Vista

Windows Presentation Foundation – WPF

WPF è il nuovo ambiente grafico di Windows Vista e .NET Framework 3.0. Il percorso formativo inizia con un'introduzione all'ambiente e alle sue caratteristiche principali, per poi entrare nel dettaglio con esempi semplici di binding di proprietà fino a esempi complessi come Master/Detail binding. Si parlerà di compatibilità con il passato, migrazione dell'esistente e interoperabilità con il futuro (tecnologia denominata "CrossBow"). Verrà poi analizzato il lato web-oriented di WPF con le Express Application, applicazioni capaci di essere ospitate all'interno di Internet Explorer per estendere le possibilità delle user interface del futuro. Brevi cenni su WPF/E (WPF Everywhere) chiudono la serie. Si tratta di tre Webcast che ti portano direttamente al cuore del nuovo sistema. Indispensabili per creare applicazioni innovative.

PERCORSI FORMATIVI

Per chi desiderasse approfondire sono disponibili sul sito di Microsoft una serie di strumenti dedicati ad approfondire le novità di Windows Vista

Windows Vista e .NET Framework 3.0: overview • Windows Communication Foundation – WCF • Windows Workflow Foundation • Windows Workflow Foundation - Security & privacy • Internet Explorer 7 • Architettura del sistema operativo • Developing Gadgets for Windows Toolbar • Come testare la compatibilità delle proprie applicazioni

Visita
http://www.microsoft.com/italy/msdn/risorsemsdn/windowsvista/default.mspx

FAQ

Cosa sono i Webcast MSDN?
MSDN propone agli sviluppatori una serie di eventi gratuiti online e interattivi che approfondiscono le principali tematiche relative allo sviluppo di applicazioni su tecnologia Microsoft. Questa serie di "corsi" sono noti con il nome di Webcast MSDN

Come è composto tipicamente un Webcast?
Normalmente vengono illustrate una serie di Slide commentate da un relatore. A supporto di queste presentazioni vengono inserite delle Demo in presa diretta che mostrano dal vivo come usare gli strumenti oggetto del Webcast

Come mai trovo riferimenti a chat o a strumenti che non ho disponibili nei Webcast allegati alla rivista?
La natura dei Webcast è quella di essere seguiti OnLine in tempo reale. Durante queste presentazioni in diretta vengono utilizzati strumenti molto simili a quelli della formazione a distanza. In questa ottica è possibile porre domande in presa diretta al relatore oppure partecipare a sondaggi etc. I Webcast riprodotti nel CD di ioProgrammo, pur non perdendo nessun contenuto informativo, per la natura asincrona del supporto non possono godere dell'interazione diretta con il relatore.

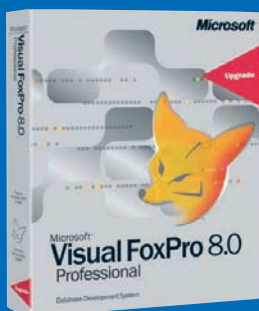
Come mai trovo i Webcast su ioProgrammo
Come sempre ioProgrammo cerca di fornire un servizio ai programmatori italiani. Abbiamo pensato che poter usufruire dei Webcast MSDN direttamente da CD rappresentasse un ottimo modo di formarsi comodamente a casa e nei tempi desiderati. Lo scopo tanto di ioProgrammo, quanto di Microsoft è infatti quello di supportare la comunità dei programmatori italiani con tutti gli strumenti possibili.

Su ioProgrammo troverò tutti Webcast di Microsoft?
Ne troverai sicuramente una buona parte, tuttavia per loro natura i webcast di Microsoft vengono diffusi anche OnLine e possono essere seguiti previa iscrizione. L'indirizzo per saperne di più è: http://www.microsoft.it/msdn/webcast/msdn segnalo nei tuoi bookmark. Non puoi mancare.

L'iniziativa sarà ripetuta sui prossimi numeri?
Sicuramente si.

News

CODICE APERTO PER VISUAL FOX PRO

È stato uno dei cavalli di battaglia di Microsoft per molto tempo ma, dopo averne cessato lo sviluppo, l'azienda di Redmond conferma adesso di volerne rendere disponibile il codice sorgente. Il passo appare logico, visto e considerato che il prodotto vanta comunque una larga base di utilizzatori. In questo modo si affida alla community ogni successivo sviluppo di Visual Fox Pro. In particolare il codice sorgente sarà rilasciato sul sito di CodePlex, comprensivo delle modifiche fin qui apportate e che dovevano costituire la base per il nuovo Visual FoxPro 10. A garanzia della volontà di MS di non lasciare abbandonata a se stessa la community degli utilizzatori, in ogni caso l'azienda di Bill Gates ne garantirà il supporto fino al 2010. Il team che fino ad ora si è occupato di FoxPro sarà invece dirottato sullo sviluppo del prossimo Visual Studio, nome in codice Orcas

ADDIO BACKUS

Per quelli che non lo conoscessero John W. Backus è considerato il padre di Fortran, il linguaggio che nel 1957 fu considerato una rivoluzione nel mondo degli sviluppatori. Backus si è spento all'età di 82 anni, ed è sorprendente quanto, nonostante il Fortran sia ormai un linguaggio relegato a pochi eletti, tuttavia la morte di Backus abbia suscitato un sincero dolore rappresentato da un numero elevatissime di persone in rete. Backus per molti anni ha anche ricoperto il ruolo di team leader in IBM e recentemente stava continuando il suo lavoro alla ricerca di modi sempre nuovi di concepire un linguaggio di programmazione

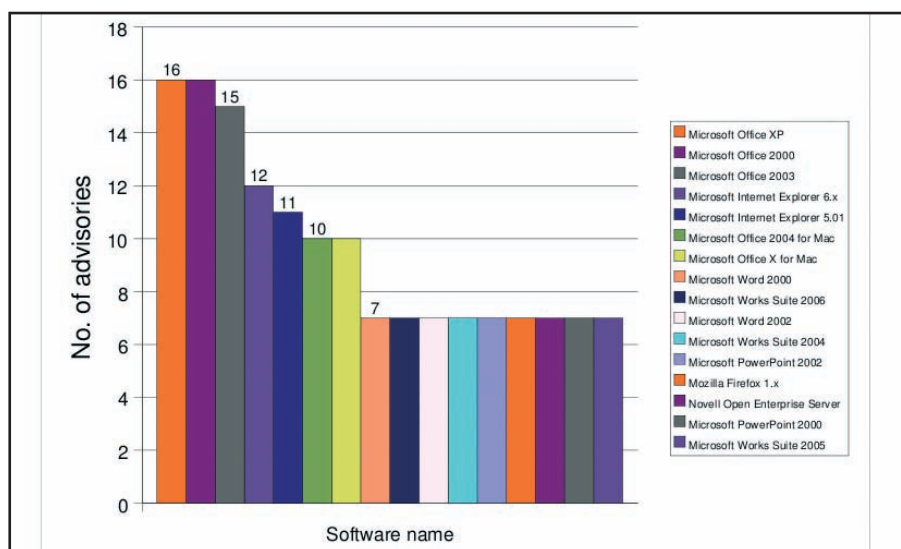
PRIMI GUAI IN VISTA

Uscito da veramente pochissimo Vista è già oggetto di discussione. Si susseguono i dati di vendita e ognuno rappresenta una realtà diversa. Fra proclami di successo e delusioni, l'unica cosa certa è comunque che cominciano ad arrivare i primi fix di sicurezza. Non si tratta, in realtà di enormi problemi attualmente. Il primo bug è relativo a Windows Mail, il software che rimpiazza Outlook. Sotto determinate condizioni sarebbe possibile inviare ed eseguire un comando remoto sulla macchina attaccata. Il bug tuttavia non rappresenta un problema enorme, di fatto non è possibile passare argomenti ai comandi in esecuzione e soprattutto il comando in questione dovrebbe essere contenuto in una cartella dal nome omonimo. Ovvio che la questione diventa piuttosto improbabile anche se non impossibile. L'altra debolezza, non può configurarsi come un problema di sicurezza, quanto invece come un bug nella programmazione del sistema. Pare infatti che su certi PC, muovendo o copiando determinati file si ottenga un rallentamento fuori controllo del sistema operativo

È IL PARADISO PER FIREFOX

Arriva la terza alpha del noto browser di Mozilla Foundation, nome in codice questa volta: "Gran Paradiso". Si tratta ancora decisamente di una versione dedicata agli sviluppatori basata sul motore di rendering Gecko 1.9. La nuova versione di Firefox è piuttosto attesa, il browser infatti nel tempo ha recuperato una posizione di rilievo nel mercato. Le modifiche di maggior spicco in questa terza alfa sono relative al supporto per le PNG animate, al supporto di alcuni attributi DOM come clientLeft e clientTop e alle modalità di caching dei dati. Il passaggio alla nuova versione non è tuttavia indolore e come sempre accade quando si parla di browser potrebbero sussistere problemi di compatibilità con la visualizzazione delle pagine sviluppate per release precedenti. D'altra parte Gecko si configura ormai come un motore affidabile a cui fa capo non solo Firefox ma tutta una serie di Browser che rappresentano un patrimonio enorme soprattutto per la crescente comunità degli utenti Linux.

SECUNIA ELEGGE IL SOFTWARE PIÙ SICURO

La nota azienda danese che si occupa di sicurezza informatica ha appena reso pubblico il rapporto 2006 sull'andamento delle vulnerabilità nei software più diffusi. Secondo questo rapporto i maggiori exploit del 2006 devono essere catalogati come "System Access" ovvero exploit che coinvolgono gli amministratori di sistema, con un incremento di quasi il 25% rispetto all'anno precedente. Secondo lo stesso rapporto il software che avrebbe sofferto delle maggiori vulnerabilità sarebbe stato Office XP, a seguire le altre versioni di Office 2000 e 2003. Quarto e quindo posto per Internet Explorer 6.0.1 ae 5.0.1. Nel complesso le prime 12 posizioni della classifica sarebbero occupate da software targato Microsoft e solo al tredicesimo e quattordicesimo software comparirebbero Mozilla Firefox 1.x e Novell Open Enterprise Server.
Questa mole di informazioni ed il relativo contenuto non deve in realtà spaventare più di tanto. Appare ovvio che essendo i software targati Microsoft quelli tipicamente più diffusi, è abbastanza giustificato il fatto che siano anche quelli che subiscono il maggior numero di attacchi e di conseguenza quelli che rilasciano più bug fix relativi alla security. Quello che invece deve far riflettere è il considerevole aumento della necessità di rilasciare bug fix per la sicurezza. Il dato dimostra la dimensione che sta assumendo ormai il mercato clandestino degli attacchi informatici. Un mercato che deve preoccupare e deve essere fronteggiato con maggiore coordinazione non solo dagli sviluppatori ma anche dalle leggi dei vari governi.

C'È ATTESA PER IL NUOVO SYMBIAN OS

Arrivano le prime indiscrezioni per la nuova versione di Symbian OS: la 9.5, prevista per il 2009. Prima di tutto la società sviluppatrice del sistema operativo ha annunciato che saranno ancora diminuiti i requisiti hardware minimi necessari a fare girare il prodotto. Si attende una riduzione dal 20 al 30% del consumo di memoria, operazione che dovrebbe aumentare quasi del 75% la velocità d'avvio delle applicazioni. Allo stesso modo il minor consumo di memoria dovrebbe aumentare notevolmente anche la durata delle batterie. Dal punto di vista strettamente operativo si prevede un maggior supporto la voice over IP e alla gestione del multimedia intesa anche come supporto a protocolli come DVB e ISDB-T. Infine c'è da segnalare il supporto nativo al database SQLLite, il che fa presupporre che la persistenza dei dati verrà adesso gestita tramite questo DB. Symbian OS, attualmente, vede un grande contributo di Nokia, società che ne detiene quasi il 48%. Allo stesso tempo il sistema operativo di Symbian detiene quasi il 65% del mercato, nonostante che Windows Mobile 5.0 continui a guadagnare posizioni. La data di rilascio prevista: il 2009, fa presupporre che una diminuzione delle richieste hardware, pur sempre gradita, non rappresenti una novità di così enorme rilievo. Di fatto due anni di evoluzione informatica rappresentano un periodo sufficientemente lungo per poter garantire che a quella data l'hardware dei dispositivi mobili sarà probabilmente ulteriormente potenziato. Interessante invece proprio in un'ottica futura il miglior supporto al Voice Over Ip e alla TV via rete

016-023 Mobile 30-03-2007 14:42 Pagina 16

AGGIORNA IL BLOG CON IL CELLULARE

IN QUESTO ARTICOLO VEDREMO COME SVILUPPARE UN'APPLICAZIONE CLIENT IN J2ME CHE SI CONNETTE AD UN SERVER IN PHP E CONSENTE DI INSERIRE DINAMICAMENTE CONTENUTI SUL WEB TRAMITE IL TELEFONINO

Utilizzando Java ME (Micro Edition), PHP e MySQL si possono sviluppare applicazioni client/server molto interessanti. In questo articolo vedremo come far interagire le tre tecnologie sviluppando una semplice e utile applicazione che ci permetterà di interagire con un blog multiutente. Il motto di Java è "Write once, run anywhere", ossia basta scrivere l'applicazione una sola volta e questa girerà su qualunque sistema. A quanto pare tale motto continua a valere anche per i dispositivi mobili. Infatti, Java ME, sin da quando è nato, ha avuto un grosso impatto nel mondo dello sviluppo orientato a cellulari e palmari. Anche se alcuni possono obiettare che Java ME non è tanto portabile quanto altre tecnologie Java, il risultato ottenuto sulla maggior parte dei dispositivi è, il più delle volte, ampiamente accettabile. Nel pensare un'applicazione client/server, dove il client è sviluppato utilizzando Java ME, la scelta più ovvia per la parte server, o meglio la prima che viene in mente, ricade su Java Enterprise Edition (EE). Ovviamente quando si ha a disposizione un web server che supporta tale tecnologia allora Java EE può rappresentare la scelta migliore. E' noto, tuttavia, che i domini che supportano JSP e Servlet non sono dei più economici! D'altro canto, acquistare un dominio che fornisce il supporto per PHP può risultare una scelta vantaggiosissima per le nostre tasche. Per tale motivo la parte server dell'applicazione oggetto dell'articolo è stata sviluppata in PHP. Ad ogni modo, se avete a disposizione un web server con supporto per Java EE, convertire gli script PHP in Servlet e/o JSP non sarà una cosa complicata. Manca ancora un tassello per completare il quadro. A parte la comunicazione client/server abbiamo bisogno di uno strumento per memorizzare, in modo permanente, le informazioni relative al blog. A tal proposito utilizzeremo un DB ed in particolare MySQL. In **figura 1** è mostrata l'interazione completa tra client (Java ME), server (PHP) e DB (MySQL).

Fig. 1: Comunicazione tra client, server e DB.

In **figura 2**, invece, potete vedere lo schema della semplice base dati utilizzata per quest'articolo. A proposito, nel codice che trovate nel CD allegato vi è uno script che potete usare per la creazione del DB in questione. Vi basterà creare un DB di nome blog e poi lanciare lo script per la creazione delle tabelle.

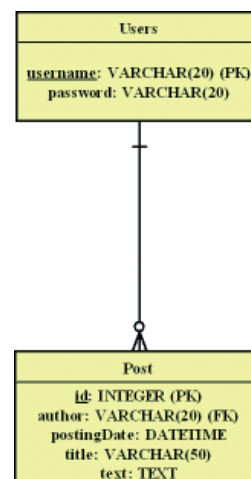



Fig. 2: Il semplice database atto a memorizzare i post e gli utenti

LA NOSTRA APPLICAZIONE

Vediamo ora in cosa consiste la nostra applicazione. Ciò che vogliamo realizzare è un software che ci permetta di gestire un blog multiutente. Per blog multiutente s'intende, in parole povere, un blog che può essere aggiornato da più persone, basta che queste abbiano un account che gli permetta di accedere, in update, al blog in questione. Come avrete intuito dalla figura 2, gli utenti con questi "poteri" di aggiornamento sono memorizzati nella tabella Users. Dato che user-

name deve essere univoco lo abbiamo scelto anche come chiave primaria della tabella. Ciò che l'utente inserisce nel blog è chiamato post. Un singolo post è formato da:

- id: la chiave primaria
- author: username dell'autore e che costituisce la chiave esterna
- posting_date: la data e l'ora di inserimento del post
- title: il titolo del post
- text: il testo del post

Chiaramente avremmo potuto complicare il nostro DB utilizzando altre tabelle per gestire ulteriori informazioni, però ciò che abbiamo visto è sufficiente per gestire un blog di base.
Andiamo, ora, ad analizzare la nostra applicazione dividendola logicamente nelle sue due parti architetturali, ossia il client ed il server.

PARTE CLIENT

Come abbiamo già accennato, la parte client dell'applicazione sarà sviluppata in Java ME. Questo ci consentirà un suo deploy su qualsiasi dispositivo mobile Java ME-compliant. Abbiamo visto che, per poter inserire un post, l'utente deve avere un account valido censito all'interno della tabella Users. Tale account sarà "mappato" localmente sul cellulare. Per tale motivo, al primo avvio dell'applicazione, viene chiesto username e password all'utente. Le credenziali dell'utente saranno memorizzate sul cellulare utilizzando RMS in modo da non doverle richiedere ogni qual volta l'applicazione viene avviata. Ecco il codice eseguito ad ogni avvio:

```
protected void startApp()
{
        // Se è il primo accesso rimanda alla form
                                        di login
        if (isLoginEmpty())
        {
                setDisplay(fmLogin);
        }
        else
        {
                setDisplay(lsMain);
        }
}
[...]
private boolean isLoginEmpty()
{
   RecordManager rm = new
            RecordManager(Constants.RMS_LOGIN);
   return (rm.numRecords() < 2);
 }
```

La prima cosa che fa è controllare se vi è un login già precedentemente effettuato. In tal caso visualizza il menu principale altrimenti forza un login e, in seguito, visualizza il menu visibile in figura 3.

Fig. 3: Menu principale dell'applicazione

RecordManager è una classe di utilità che ho scritto per semplificare le operazioni di scrittura, modifica, lettura ed eliminazione di record dal record store. RecordManager incapsula alcune API di RMS in modo da "simulare" una memorizzazione diretta di stringhe e non di array di byte, il che è molto più comodo per i nostri scopi. Nel record store riguardante il login, username è memorizzato nel primo record mentre la password nel secondo. Se il numero di record è inferiore a due significa che il record store creato è vuoto e quindi non vi sono ancora le credenziali dell'utente.
Le voci del menu principale sono:

- Scrivi Post
- Upload Post
- Download Post
- Logout

In particolare *Logout* elimina le credenziali dal re-

RISORSE UTILI

Sun Java Wireless Toolkit for CLDC:
http://java.sun.com/products/sjwtoolkit/
PHP:
http://www.php.net
MySQL:
http://www.mysql.com
Apache HTTP Server:
http://httpd.apache.org/

cord store e richiede le nuove. Questo permetterà all'utente un'eventuale modifica dei propri dati oppure l'utilizzo dell'applicazione da parte di qualche altro utente.

SCRITTURA DI UN POST

Selezionando Scrivi Post dal menu principale l'utente vedrà un form simile a quello di figura 4.

Fig. 4: Form che consente la scrittura di un nuovo post

Inserendo titolo e testo e scegliendo *Memorizza* dalla lista dei comandi (*Menu*), il post viene aggiunto alla lista locale. Abbiamo scelto di memorizzare i post localmente ed effettuarne l'upload in un colpo solo per ottimizzare gli accessi alla rete. Selezionando, poi, *Upload Post* dal menu principale, tutti i post presenti nel record store saranno trasferiti al server.
Per creare un form abbiamo esteso la classe Form ed implementato l'interfaccia CommandListener, entrambe del package javax.microedition.lcdui. In particolare l'implementazione del metodo commandAction di CommandListener ci permette di gestire i comandi selezionati dall'utente. Ecco l'inizio della classe FormWritePost:

```
public class FormWritePost extends Form implements
                                        CommandListener
{
        // midlet
        private MidletBlog midlet;
        // display precendente
        private Displayable previousDisplay;
        // item
        private TextField tfTitle;
        private TextField tfText;
        // comandi
        private static final Command cmExit =
                        CommandBuilder.getExit();
        private static final Command cmStore =
                        CommandBuilder.getStore();
        private static final Command cmBack =
                        CommandBuilder.getBack();
        [...]
```

Come si può vedere, vi è un riferimento alla midlet ed uno al display precedente. In particolare, quest'ultimo lo utilizziamo quando viene scelto il comando Indietro. In seguito, vi sono due textbox atti a ricevere titolo e testo del post. Il codice restante concerne i tre comandi riguardanti il form, ossia: *Esci*, *Memorizza* ed *Indietro*. Il primo termina l'applicazione. Il secondo memorizza il post usando RMS. Il terzo torna al menu principale. I vari comandi utilizzati in tutta l'applicazione sono creati attraverso la classe CommandBuilder che è una sorta di "command provider". Quello che segue è, invece il codice concernente il costruttore della classe in questione:

```
public FormWritePost(MidletBlog midlet, Displayable
                                        previousDisplay)
{
        super("Scrivi Post");
        this.midlet = midlet;
        this.previousDisplay = previousDisplay;

        tfTitle = new TextField("Titolo", "", 50,
                                        TextField.ANY);
        tfText = new TextField("Testo", "", 500,
                                        TextField.ANY);
        // Aggiunge gli item al form
        append(tfTitle);
        append(tfText);
        // Aggiunge i comandi
        addCommand(cmExit);
        addCommand(cmStore);
        addCommand(cmBack);

        // Imposta il listener
        setCommandListener(this);
}
```

NOTA

IDE PER JAVA ME

Per sviluppare applicazioni con Java Micro Edition potete tranquillamente utilizzare il Wireless Toolkit della Sun. Se, tuttavia, siete degli appassionati di Eclipse potreste voler provare un plug-in apposito, ossia EclipseME che trovate al seguente indirizzo: http://eclipseme.org/ Io lo uso regolarmente e lo trovo abbastanza buono.

Esso invoca il costruttore di Form passando il titolo da visualizzare. In seguito valorizza midlet e previousDisplay, istanzia i textbox e li aggiunge, assieme ai comandi, al form vero e proprio. L'ultima riga di codice serve ad impostare il listener dei comandi per il form appena creato. Quando l'utente sceglie un comando viene invocato il metodo commandAction che è l'unico dichiarato nell'interfaccia CommandListener e che dobbiamo implementare per rispettarne il contratto. Eccone il codice:

```
public void commandAction
                    (Command c, Displayable d)
{
        if (c == cmExit)
        {
                midlet.shutDownApp(false);
        }
        else if (c == cmBack)
        {
        midlet.setDisplay(previousDisplay);
        }
        else if (c == cmStore)
        {
                // titolo
                String title = tfTitle.getString();
                // testo
                String text = tfText.getString();
                String post = title +
                   Constants.SEPARATOR + text;
                // Memorizza il record
                RecordManager rm = new
           RecordManager(Constants.RMS_POSTS);
                if (rm.addRecord(post))
                {
                        midlet.showInfo("Post
           memorizzato correttamente", this);
                        tfTitle.setString("");
                        tfText.setString("");
                }
                else
                {
                        midlet.showInfo(
"Post non memorizzato correttamente, riprova più
                                   tardi", this);
                }
        }
}
```

Il metodo commandAction prende due parametri. Il primo è un riferimento al comando scelto dall'utente. Il secondo indica su quale oggetto Displayable è stato selezionato il comando. Nel nostro caso esso è il form vero e proprio. La parte di codice più interessante è quella riguardante la selezione del comando cmStore, ossia quello concernente la memorizzazione del post. Il blocco di codice relativo, recupera titolo e testo dalle textbox e costruisce la stringa che rappresenta un post. SEPARATOR è la stringa da utilizzare per separare logicamente titolo e testo del post. Nel nostro caso abbiamo usato il carattere | (pipe). Ovviamente potete cambiarlo a seconda le vostre necessità. SEPARATOR, assieme ad altre costanti, è memorizzato nella classe Constants.
Il codice continua con la memorizzazione del post attraverso RMS. Come potete notare, anche in questo caso ci torna utile la classe RecordManager. In sostanza viene memorizzato il post e svuotati i campi titolo e testo per consentire l'aggiunta di ulteriori post. Se tutto va a buon fine viene notificato il successo altrimenti un fallimento. Tutto ciò servirebbe a ben poco se non ci fosse una funzionalità che permette di inviare i post al cellulare.

UPLOAD DEI POST

Quando, dal menu principale, viene selezionato Upload Post la midlet esegue il seguente codice:

```
// Recupera i post usando RMS
RecordManager rm = new
        RecordManager(Constants.RMS_POSTS);
String[] posts = rm.getAllRecords();
// Invia i post
PostSender ps = new PostSender(this, lsMain, new
                    Login().getUser(), posts);
ps.send();
```

Tale codice recupera, come array di stringhe, tutti i post precedentemente memorizzati. In seguito, attraverso la classe PostSender, li invia al server. Il costruttore di tale classe è il seguente:

```
public PostSender(MidletBlog midlet, Displayable
                             previousDisplay,
              User user, String[] posts)
{
        this.midlet = midlet;
        this.previousDisplay = previousDisplay;
        this.user = user;
        this.posts = posts;
}
```

Esso prende in ingresso quattro parametri, che sono: il riferimento alla midlet, il riferimento al display precedente, un riferimento di tipo User ed un array di stringhe che rappresentano, appunto, i post da inviare. Tra i campi privati della classe vi è anche l'URL da utilizzare:

```
private static final String URL =
 "http://localhost/MobileBlogServer/post_sender.php";
```

Esso fa riferimento allo script post_sender.php che, come vedremo quando analizzeremo la parte server,

si occupa della persistenza dei post. Il metodo send della classe PostSender invia, in un thread separato, i post al server. L'avvio di tale thread invoca il metodo responsabile dell'invio vero e proprio. Tale metodo è sendPost:

```
private void sendPost() throws IOException
{
        String agent = "Mozilla/4.0";
        String type = "application/x-www-form-
                                    urlencoded";
        // Costruisce il corpo della richiesta da
                                          inviare,
        // ossia il payload
        String qs = buildRequestBody();

        InputStream is = null;
        HttpConnection http = null;
        DataOutputStream dos = null;
        StringBuffer sb = new StringBuffer();
        try
        {
                // Stabilisce la connessione
                http = (HttpConnection)
        Connector.open(URL, Connector.READ_WRITE);
                // Imposta il metodo come POST
http.setRequestMethod(HttpConnection.POST);
                // Imposta gli altri header
                http.setRequestProperty
                          ("User-Agent", agent);
http.setRequestProperty("Content-Type", type);
http.setRequestProperty("Content-Length", "" +
                                    qs.length());
                // Invia il payload
                dos =
                   http.openDataOutputStream();
                byte[] byteRequest =
                                   qs.getBytes();
                for (int i = 0; i <
                          byteRequest.length; i++)
                {
dos.writeByte(byteRequest[i]);
                }
                // Riceve il response
                is = new
        DataInputStream(http.openInputStream());
                int ch;
                while ((ch = is.read()) != -1)
                {
                        sb.append((char) ch);
                }
        }
        catch (Exception e)
        {
                System.err.println("Error: " +
                                     e.toString());
                networkError(e.toString());
        }
        [...]
}
```

La parte più interessante di questo codice è l'invio delle informazioni al server tramite il metodo HTTP POST. Non sarebbe stato praticabile inviare i dati tramite GET dato che i post possono avere una dimensione notevole. Come si può notare, quel codice, dopo aver aperto la connessione, imposta il metodo a POST ed i vari header necessari per utilizzare tale metodo. In seguito invia il payload, ossia le informazioni necessarie, utilizzando un DataOutputStream. Il payload lo abbiamo ottenuto grazie alla chiamata al metodo buildRequestBody:

String qs = buildRequestBody();

Il codice di tale metodo è il seguente:

```
private String buildRequestBody()
{
        StringBuffer ret = new StringBuffer();
        // username
        ret.append("username=" +
             Utils.encodeURL(user.getUsername()));
        // password
        ret.append("&password=" +
              Utils.encodeURL(user.getPassword()));
        // separatore
        ret.append("&separator=" +
          Utils.encodeURL(Constants.SEPARATOR));
        // messaggio di successo
        ret.append("&success=" +
   Utils.encodeURL(Constants.SUCCESS_RESPONSE));
        // Aggiunge i post nel formato:
       post[]=titolo1|testo1&post[]=titolo2|testo2...
        for(int i = 0; i < posts.length; i++)
        {
                ret.append("&post[]=" +
                         Utils.encodeURL(posts[i]));
        }
        return ret.toString();
}
```

In pratica viene creata una stringa che ha il formato di una query string, ossia:

```
key1=value1&key2=value2...
```

Tale stringa contiene username, password, il carattere usato per separare titolo e testo del post, il messaggio di successo ed i post veri e propri. Abbiamo scelto di inviare il carattere separatore al server di modo che, se dovessimo decidere di cambiare questo carattere in futuro, il lato server continuerebbe a funzionare senza problemi. Per quanto riguarda il messaggio di successo, la necessità di gestirlo lato client sta nel fatto che, se l'upload va a buon fine, dobbiamo

NOTA

EASYPHP

Se non siete pratici dell'installazione di PHP, Web server e MySQL allora potete utilizzare EasyPHP. Installando quest'ultimo otterrete, in un colpo solo, Apache Web Server, MySQL, PHP e PHPMyAdmin. Quest'ultimo, in particolare, vi permetterà di creare il DB molto agevolmente attraverso una comoda interfaccia. L'indirizzo di riferimento è il seguente: http://www.easyphp.org/?lang=it

svuotare il record store locale dove risiedono i post. Capiamo che l'upload è andato a buon fine nel momento in cui il server ci torna indietro il messaggio di successo da noi inviato.
Da notare, inoltre, che i vari post sono inviati utilizzando la seguente sintassi:

```
post[]=post1&post[]=post2...
```

Questo ci consentirà di ricevere lato server, come vedremo, i vari post in un unico array. Il resto del codice non richiede particolari spiegazioni. Dopo l'invio del post si attende il response del server e viene mostrato un alert sia in caso di successo sia di fallimento. Vi è da sottolineare che in caso di successo il record store rappresentante i post viene svuotato.

DOWNLOAD DEL POST

Selezionando *Download Post* dal menu principale si avvia il thread responsabile del download dell'ultimo post dal server. Il codice responsabile di ciò è il seguente:

```
PostRetriever pr = new PostRetriever(this, lsMain);
pr.retrieve();
```

La classe PostRetriever, dal punto di vista "didattico", è simile alla precedente. L'unica differenza sostanziale è che la richiesta viene effettuata utilizzando GET invece di POST. Inoltre, l'URL utilizzato è il seguente:

```
private static final String URL =
        "http://localhost/MobileBlogServer/post_
                                    retriever.php";
```

Il metodo che, internamente, viene invocato dal thread è getPost. Eccone il codice:

```
private void getPost() throws IOException
{
        InputStream is = null;
        StringBuffer sb = null;
        HttpConnection http = null;
        try
        {
                String completeURL = URL +
                                    "?separator=" +
                Utils.encodeURL(Constants.SEPARATOR);
                // Stabilisci la connessione
                http = (HttpConnection)
                        Connector.open(completeURL);
                // Imposta GET come metodo
                                          HTTP
http.setRequestMethod(HttpConnection.GET);
                // Gestisci il response
                if (http.getResponseCode() ==
HttpConnection.HTTP_OK){
                        sb = new
                                StringBuffer();
                        int ch;
                        is =
                        http.openInputStream();
                        while ((ch = is.read())
                                        != -1)
sb.append((char) ch);
                }
                else
                {
System.out.println("Network error");
                        networkError();
                }
        }
        catch (Exception e)
        {
                System.err.println("Error: " +
                                e.toString());
                networkError(e.toString());
        }
        finally
        {
                [...]
        }
        if (post != null)
        {
                System.out.println(post);
                midlet.showPosts(post,
                        getCurrentDisplay());
        }
        else
        {
                networkError();
        }
}
```

Come si può vedere l'utilizzo di GET è molto semplice rispetto a POST. In tal caso l'impiego di tale metodo è più che giustificato dato che l'URL della richiesta comprensivo di query string ha dimensioni ragionevoli per una richiesta di tipo GET. Se tutto è andato a buon fine il post scaricato viene visualizzato tramite il metodo showPosts della classe MidletBlog, ossia l'unica midlet dell'applicazione. Il codice di showPosts è il seguente:

```
public void showPosts(String rawPost, Displayable d)
{
        PostParser pp = new PostParser(rawPost);
        try
        {
                // Estrae il post dalla stringa in
                                formato "grezzo"
                Post post = pp.parse();
                // Lo visualizza
                FormDisplayPost fdp = new
```

COSA VUOL DIRE RMS?

RMS sta per Record Management System. Esso è il sistema utilizzato da Java ME per la persistenza di informazioni. Viste le capacità hardware ridotte dei cellulari non ci si poteva aspettare di trovare Oracle o qualsiasi altro DB installato su di esso! In Java ME le informazioni sono, quindi, memorizzate all'interno di Record Store. Un record store è formato da un insieme di record dove ogni record è visto come un array di byte. Ogni MIDlet può gestire più record store. La nostra applicazione ne gestisce due. Uno relativo alla memorizzazione delle credenziali e l'altro concernente la persistenza temporanea dei post.

```
                    FormDisplayPost(this, d, post);
            setDisplay(fdp);
        }
        catch (IllegalArgumentException iae)
        {
            [...]
        }
[...]
}
```

Il post scaricato è in una forma "grezza", ossia è formato da titolo e testo separati dal carattere separatore che abbiamo già visto. Per tale motivo viene utilizzato PostParser per eseguire il parsing del post. Il metodo parse spezza la stringa che rappresenta il post nelle varie parti e restituisce un oggetto di tipo Post costruito tramite questi token. La visualizzazione vera e propria del post avviene tramite il form FormDisplayPost che non vedremo per brevità.

PARTE SERVER

La parte server, oltre al database, è costituita da tre file PHP: *config.inc.php*, *post_sender.php e post_retriever.php*. Il primo è un file incluso dagli altri due; esso contiene le configurazioni riguardanti il DB, la funzione connect che effettua la connessione vera e propria ed una funzione di utilità che controlla l'invio dei parametri obbligatori.
L'invio dei post avviene attraverso *post_sender.php*. Ecco l'inizio dello script:

```
<?php
header("Content-type: text/plain");
require("config.inc.php");

$required_parameters = array("username",
            "password", "separator", "success", "post");
check_parameters($required_parameters);

// recupera i dati inviati dal client
$username =
        trim($_REQUEST[$required_parameters[0]]);
$password =
        trim($_REQUEST[$required_parameters[1]]);
$separator =
        trim($_REQUEST[$required_parameters[2]]);
$success =
        trim($_REQUEST[$required_parameters[3]]);

$posts = $_REQUEST[$required_parameters[4]];
[...]
```

Come si può vedere, tale script imposta il content type del response a text/plain dato che esso sarà una mera notifica di successo o fallimento della scrittura dei post nel DB. Il file *config.inc.php* contiene, come abbiamo detto, le funzioni check_parameters e connect, utilizzate per controllare l'invio dei parametri e connettersi al DB, rispettivamente. I parametri che lo script si aspetta sono: username dell'utente, la sua password, il carattere utilizzato come separatore di titolo e testo dei post, il messaggio di successo ed i post veri e propri inviati come array di stringhe. Il codice continua come segue:

```
// connessione al DB
$db = connect();
// imposta la query da usare per controllare le
                                        credenziali
$query = "SELECT username, password FROM users
            WHERE username='$username' AND
                        password='$password'";
// la esegue
$result = mysql_query($query, $db);

if(mysql_num_rows($result) == 0)
{
        mysql_close($db);
        // credenziali non valide
        echo "Username e/o password non validi";
        die();
}
```

Questo pezzo di codice effettua la connessione al DB e controlla che le credenziali inviate dall'utente siano valide. In caso negativo chiude la connessione col DB, restituisce al client (al cellulare) una notifica ed interrompe lo script. Se le credenziali, invece, sono valide il codice eseguito è il seguente:

```
else // credenziali valide
{
        $error = false;

        // inserisce nel DB tutti i post ricevuti
        foreach($posts as $post)
        {
                $pieces = explode($separator,
                                        $post);
                $title = addslashes($pieces[0]);
                $text = addslashes($pieces[1]);
                // data odierna
                $date = date("Y-m-d H:i:s");
                // imposta la query
                $query = "INSERT INTO
                posts(author, posting_date, title, text)
        VALUES('$username', '$date', '$title', '$text')";
                // la esegue
                $result = mysql_query($query,
                                        $db);

                if(!$result)
                {
                        $error = true;
```

```
                        break;
                }
        }
        if($error)
        {
                echo "Non tutti i post sono stati
          inseriti correttamente. Riprova più tardi";
        }
        else
        {
                echo $success;
        }
}
mysql_close($db);
?>
```

Il codice precedente non fa altro che ciclare sui post ricevuti ed inserirli, uno ad uno, nel DB. Se si verifica un errore durante tale operazione il ciclo si interrompe ed il client viene notificato del fallimento. Se tutto va a buon fine, invece, al client viene inviato il messaggio di successo che esso stesso ci ha richiesto di restituire in caso di successo. Infine viene rilasciata la connessione precedentemente acquisita.
L'operazione complementare consiste nello scaricare, sul cellulare, l'ultimo post inserito. Quest'operazione potrebbe sembrare senza senso nel caso in cui il blog sia monoutente. Dato che, tuttavia, il blog è multiutente potrebbe aver senso sapere chi è stato l'ultimo a postare qualcosa e conoscere il contenuto del post. Lo script lato server atto a recuperare l'ultimo post è *post_retriever.php*. Il suo codice è il seguente:

```
<?php
require("config.inc.php");
$required_parameters = array("separator");
check_parameters($required_parameters);
$separator =
        trim($_REQUEST[$required_parameters[0]]);
// connessione al DB
$db = connect();
// imposta la query da usare per recuperare l'ultimo
                                              post

$query = "SELECT author, posting_date, title, text
        FROM posts ORDER BY id DESC LIMIT 0,1";
//execute it
$result = mysql_query($query, $db);
if(mysql_num_rows($result) == 0)
{
        echo "Nessun post presente";
        die();
}
else
{
        $rawPost = "";
        while(list($author, $posting_date, $title,
                $text) = mysql_fetch_array($result))
        {
                $text = stripslashes($text);
                // mette il post nel formato
                        autore|data|titolo|testo
                $rawPost .= $author .
                                        $separator .
                $posting_date . $separator .
                $title . $separator .
                $text;
        }
        echo $rawPost;

}
mysql_close($db);
?>
```

In questo caso l'unico parametro richiesto è il carattere utilizzato come separatore. Questo perché il client, come già visto, si aspetta il post nel formato: autore|data|titolo|testo, supposto che il separatore usato sia il carattere pipe (|). Questo carattere, come già detto, può essere modificato a piacimento ed è "comandato" dal client. In pratica stiamo dicendo al server: "Recuperami l'ultimo post e mettimi i vari campi in una stringa unica separandoli tramite il carattere separatore che ti ho inviato nella richiesta". Lo script quindi esegue la query, mette il risultato nel formato desiderato e lo restituisce al client. Da notare la query utilizzata per ottenere l'ultimo post:

```
SELECT author, posting_date, title, text FROM posts
                        ORDER BY id DESC LIMIT 0,1
```

In pratica, ordinando per id in modo decrescente ed utilizzando la clausola LIMIT 0,1 otteniamo l'ultimo post in ordine di inserimento, che è proprio quello che volevamo.

CONCLUSIONI

In quest'articolo abbiamo visto come sviluppare una completa applicazione client/server dove il client è costituito da un comune cellulare che supporta Java ME ed il server da un classico web server con supporto per PHP. In più abbiamo fatto uso di un DB, nel nostro caso MySQL ma potete usare quello a voi più consono. Ovviamente l'applicazione è ampiamente costumizzabile ed ampliabile a piacimento. Ad esempio, potreste voler scaricare più post invece che uno solo e visualizzare tale lista. Un'altra miglioria sarebbe gestire l'inserimento dei post con una transazione di modo che vengano inseriti o tutti o nessuno. Chiaramente lo sviluppo di queste funzionalità è lasciato come esercizio al lettore.

Alessandro Lacava

SVILUPPARE GADGET PER WINDOWS VISTA

IN QUESTO ARTICOLO IMPARAREMO AD USARE UNA DELLE TECNOLOGIE PIÙ DIVERTENTI LEGATE ALLA NUOVA VERSIONE DI WINDOWS. PARLEREMO DELLA SIDEBAR E SVILUPPEREMO UNA "MINIAPPLICAZIONE" CHE MONITORA LO SPAZIO DISPONIBILE SULL'HARD DISK

Ok, cosa è un gadget per Windows Vista? Togliamoci subito il pensiero e diciamo che si tratta di una miniapplicazione visibile nella sidebar. D'accordo non avete installato Windows Vista, per cui il termine SideBar non vi dice assolutamente niente di nuovo. Sostanzialmente la SideBar di Windows Vista si può considerare come una sorta di contenitore adagiato sul Desktop. Tipicamente all'interno di questo contenitore vivono delle miniapplicazioni: i gadget. Nell'immagine di seguito potete vedere appunto un desktop di Windows Vista, corredato di SideBar all'interno della quale ci sono tre gadget. Il primo mostra costantemente l'ultima copertina di una delle riviste prodotte dalla Edizioni Master: Win Magazine, il secondo mostra un orologio con l'ora corrente, il terzo mostra uno slideshow di immagini random prelevate dalla rete.

Figura 1: Il desktop di vista. Sul lato destro e ben visibile la sidebar

Inutile dire che il Gadget che mostra l'ultima copertina di Win Magazine è stato creato dai ragazzi di ioProgrammo, mentre gli altri due si trovano già inclusi nell'installazione di default di Windows Vista. In questo articolo impareremo appunto a sviluppare un gadget per Windows Vista.

DUE PAROLE SUI GADGET

Prima di cominciare sarà bene fare un po' di esperienza sugli oggetti che andremo ad utilizzare. Il sito http://microsoftgadgets.com/Gallery/ contiene una lista lunghissima di gadget da poter prelevare gratuitamente ed installare sul vostro computer. L'installazione è davvero banale, è sufficiente scovare sul web il gadget che vi interessa, cliccare, ed il gioco è fatto! Ve lo ritrovate per magia nella sidebar. Questo non è ovviamente l'unico modo per installare un Gadget. Qualcuno può mandarvene uno via email ad esempio, ciò che conta è che se trovate un file con estensione .gadgets sarà sufficiente cliccarci sopra due volte per ritrovarvelo, bello come il sole, nella vostra sidebar. Se avrete la pazienza di provare qualche Gadget fra quelli esistenti, scoprirete che le possibilità di interazione sono tante, di fatto è possibile usare form, input immesso dall'utente, database, fetch di informazioni dal Web e così via. Inutile dire che sui gadget si sta sviluppando un mercato enorme e che tutti i più grossi produttori di software stanno cominciando a sviluppare le proprie applicazioni agganciandovi spesso un gadget, motivo per cui noi programmatori non possiamo non imparare a programmarne uno.

DENTRO LA TECNOLOGIA

Al solito, il miglior modo per imparare una nuova tecnica è metterci le mani dentro. È esattamente quello che faremo in questo paragrafo.

1) Abilitate la sidebar: potete farlo dal menu accessori di Windows Vista cliccando su "Win-

dows Sidebar”

2) **Avviate un prompt di Windows** e al suo interno digitate “%userprofile%\AppData\Local\Microsoft\Windows Sidebar\Gadgets”

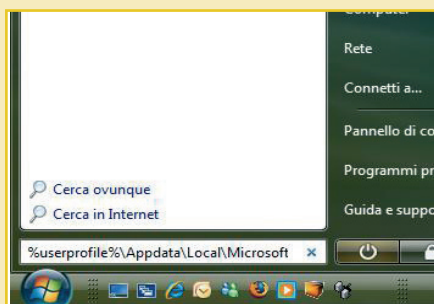
3)Si aprirà una cartella, qui dentro vanno posizionati tutti i gadget relativi all’utente che state utilizzando

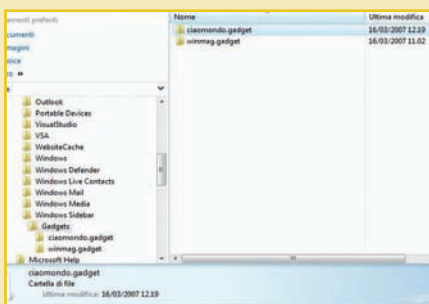
4)Create qui dentro una cartella chiamata ciaomondo.gadget

L’infrastruttura per il nostro primo gadget è

IL NOSTRO PRIMO GADGET IN TRE PASSI

UNA V OLTA CREATA LA STRUTTURA DI BASE, TUTTO VIENE DELEGATO ALLA NOSTRA FANTASIA

> IL GIUSTO PERCORSO

1 Apri il prompt dei comandi e portati nella directory dell’utente che conterrà il gadget

> I NOMI DELLE CARTELLE

2 Crea qui dentro una cartella nome.gadgets, e metti qui dentro il file manifest e gli altri

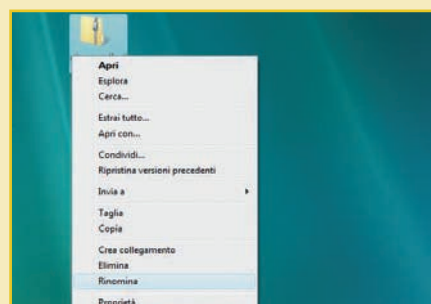
> LA DISTRIBUZIONE

3 Zippa la cartella e rinomina l’estensione da .zip a .gadgets, e sei pronto per distribuire

Figura 4: il gadget non appare centrato

pronta. Da questi primi 4 passi devono risultare chiare solo un paio di cose: tutti i gadget relativi ad un certo utente vanno nella cartella a cui ci si riferisce con il passo 2. I file necessari alla costruzione del gadget vanno in una sottocartella di quella precedente nominata con il nome del gadget e con l'estensione .gadget

QUALCHE FILE

È il momento di costruire fisicamente il nostro gadget. D'accordo, potete meravigliarvi quanto vi pare, ma diciamo subito che non c'è necessità di creare un'eseguibile né di avventurarsi all'interno di complicatissime chiamate di sistema. Un gadget è costituito da una serie di file HTML, qualche XML, CSS se vi servono e, se proprio volete divertirvi, un po' di JavaScript. Abbastanza stupiti? Non ancora? Ok vediamo cosa dobbiamo mettere nella nostra cartella ciaomondo.Gadget.

1) Un file XML che descriverà il gadget. Questo file viene chiamato anche manifest del gadget. Al suo interno verranno definite cose come il nome del gadget, l'icona con cui deve apparire nel pannello di controllo, un'eventuale descrizione

2) Un file HTML che conterrà la logica del gadget. Ovvero la sua interfaccia grafica, gli script javascript e tutto quello che serve per "far vivere" il nostro gadget. Ovviamente se siete dei programmatori provetti creerete una directory per contenere i CSS, una per i Javascript etc, ed includerete tutto nel file HTML che invece definirà solo l'interfaccia

3) Immagini, script, file CSS che possono servire al gadget

4) Un'icona che rappresenterà il gadget nel pannello di controllo.

IL FILE MANIFEST

All'interno della cartella ciaomondo.gadget create un file chiamato gadget.xml e riempitelo con le seguenti istruzioni:

```
<?xml version="1.0" encoding="utf-8" ?>
<gadget>
  <name>Ciao Mondo</name>
 <namespace>ioProgrammo.samples
                                   </namespace>
  <version>1.0</version>
  <author name="Fabio Farnesi">
    <info url="forum.ioprogrammo.it" />
  </author>
  <copyright>2007</copyright>
  <description>ioProgrammo - Developer Italian
                                   Style</description>
  <hosts>
    <host name="sidebar">
      <base type="HTML" apiVersion="1.0.0"
                          src="ciaomondo.html" />
      <permissions>full</permissions>
      <platform minPlatformVersion="0.3" />
    </host>
  </hosts>
</gadget>
```

Non entriamo attualmente nel dettaglio del senso delle singole righe, ma in linea generale se avrete la pazienza di dare uno sguardo al file, concorderete che è abbastanza intuibile che in questo manifest vengono "dichiarate" le caratteristiche essenziali di questo gadget. Tra le altre cose viene anche dichiarato che il file che contiene il "core" del gadget sarà "ciaomondo.html".

IL FILE CIAOMONDO

Creiamo un file ciaomondo.html allo stesso livello del manifest riempiendolo con le seguenti istruzioni:

```
<html>
<head>
    <title>Ciao Mondo</title>
 </head>
<body>
    <span id="gadgetContent">Ciao
                            Mondo</span>
</body>
</html>
```

A questo punto il nostro primo gadget è pronto. Non ci resta che cliccare sull'icona

per accedere al pannello di controllo dei widgets.
e proseguire con l'installazione.
È facile notare che il nostro nuovo widget compare fra quelli disponibili. Tuttavia cliccando due volte l'effetto sarà solo quello parzialmen-

te desiderato. Il widget comparirà nella Side-Bar ma l'etichetta ciao mondo sarà troppo spostata rispetto allo schermo ed inoltre lo sfondo non sarà trasparente. Esattamente come in **figura 4.**

OTTIMIZZAZIONI

Iniziamo con il referenziare all'interno di ciamondo.html un file .css contenente il foglio di stile per il gadget, come segue:

```
<html>
<head>
    <title>Ciao Mondo</title>
    <link href="styles/styles.css" type="text/css"
                                  rel="stylesheet" />
 </head>
<body>
    <span  id="gadgetContent">Ciao
                                       Mondo</span>
</body>
</html>
```

e ovviamente creiamo il corrispondente foglio di stile styles/styles.css

```
BODY
{
          width:125;
          height:150;
          padding-top: 5px;
}
#gadgetContent
{
          font-family: Tahoma;
          font-size: 10pt;
}
```

Adesso il risultato è decisamente accettabile. Il

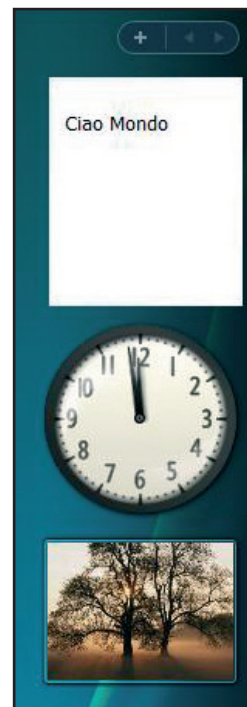



Figura 5: Il gadget appare centrato dopo averlo sistemato con i fogli di stile

nostro gadget appare centrato **figura 5**. Tuttavia non abbiamo ancora ottenuto l'effetto trasparenza. Questa è la parte più semplice. E' sufficiente creare una qualunque immagine trasparente al 100% e utilizzarla come background del file HTML. Possiamo referenziarla nel foglio di stile come segue:

```
BODY
{
        width:125;
        height:150;
        padding-top: 5px;
        background:
                url('../images/Background.png');
}

#gadgetContent
{
        font-family: Tahoma;
        color: White;
        font-size: 10pt;
}
```

Abbiamo anche cambiato il colore del font per ottenere un effetto consistente. Il nostro primo gadget è terminato! Il passo "Ciao Mondo" è compiuto, siamo ora pronti per qualcosa di più entusiasmante.

IL MODELLO AD OGGETTI

In realtà quello che abbiamo fatto fino ad ora è stato scrivere una pagina HTML, è evidente che all'interno di questa pagina posso convivere tutti gli elementi utilizzabili in IE7. Parliamo dunque, di FLASH, ActiveX , oggetti di sistema richiamabili da IE. Ad esempio modifichiamo il file ciamondo.html come segue

```
<html>
<head>
    <title>Ciao Mondo</title>
    <link href="styles/styles.css" type="text/css"
                    rel="stylesheet" />

    <script type="text/javascript"
                src="scripts/iop.js"></script>

 </head>
<body onload="setContentText()">
    <span id="gadgetContent">Ciao Mondo</div>
</body>
</html>
```

Abbiamo cioè incluso un riferimento al file di script iop.js e agganciato all'evento onload della pagina la funzione setContentText() in esso contenuta. Il file iop.js potrebbe contenere qualcosa del genere:

```
// JScript source code
function setContentText() {
   fso = new ActiveXObject
               ( "Scripting.FileSystemObject" );
   e = new Enumerator( fso.Drives );
   gadgetContent.innerHTML='';
   for(; !e.atEnd(); e.moveNext() )
   {
      drive = e.item();
      if ( drive.IsReady )
      {
         var total = Math.round
                ( drive.TotalSize / 1048576 );
         var free = Math.round
               ( drive.FreeSpace / 1048576 );
         var used = Math.round(total - free);
         gadgetContent.innerHTML
                +=drive.DriveLetter+':<br>';
         gadgetContent.innerHTML+='Spazio
                  Totale:<br> '+total+'<br>';
         gadgetContent.innerHTML+='Spazio
                  Libero:<br> '+free+'<br>';
         gadgetContent.innerHTML+='Spazio
                  Usato:<br> '+used+'<br>';
         gadgetContent.innerHTML+=
                           'Percent. usata:
            '+Math.round(( used / total)*100)+'%';
      }
```

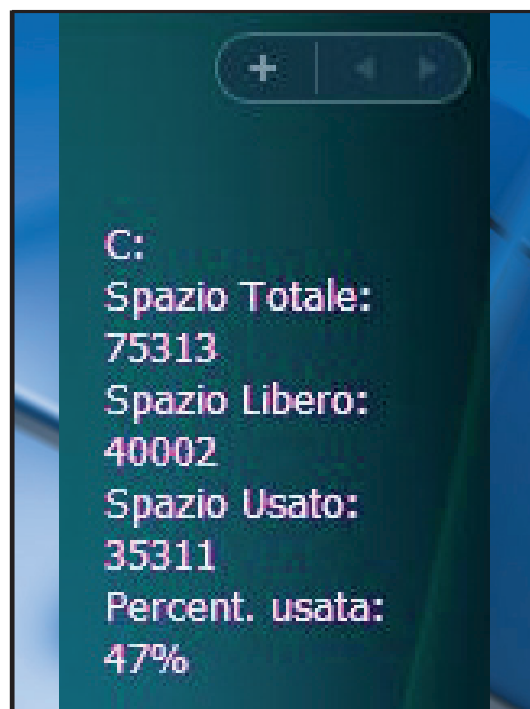



***Figura 6:** le informazioni vengono mostrate nel gadget*

```
  }
}
```

Abbiamo cioè usato gli oggetti di sistema per reperire le informazioni relative allo spazio disponibile e usato sui nostri HD e le abbiamo mostrate nel gadget.
Il risultato è quello in **figura 6**.
Non è certo graficamente bellissimo, ma non siamo qui per fare un corso di grafica e sicuramente in molti di voi sapranno personalizzarlo nel migliore dei modi. Ciò che è importante capire è che è possibile usare l'intero modello ad oggetti disponibile per IE7.

E LE OPZIONI?

Di solito gli utenti amano personalizzare i propri gadget, e d'altra parte deve esistere un qualcosa per poter inserire delle opzioni. Supponiamo ad esempio che l'utente voglia che il colore del testo del gadget che abbiamo appena appena sviluppato sia rosso invece che bianco. Come fare a interagire con un gadget? Ci viene in aiuto il metodo SettingsUI. Modifichiamo il file ciaomondo.html come segue:

```
<html>
<head>
    <title>Ciao Mondo</title>
    <link href="styles/styles.css" type=
                        "text/css" rel="stylesheet" />
    <script type="text/javascript"
                        src="scripts/iop.js"></script>
      <script type="text/javascript">
        System.Gadget.settingsUI="settings.html";
System.Gadget.onSettingsClosed=SettingsClosed;
        function SettingsClosed() {
colore=System.Gadget.Settings.read("colore");
            setContentText();
        }
    </script>

  </head>
<body onload="setContentText()">
    <span id="gadgetContent">Ciao
                                        Mondo</span>
</body>
</html>
```

Ok, ora tenetevi forte, il meccanismo è semplicissimo, ma entreremo in una girandola di file perciò abbiate la pazienza di seguire punto per punto.

1) Con l'istruzione System.Gadget.settingsUI="settings.html"; otteniamo due risultati. Prima di tutto, da ora in poi, ogni volta che il gadget verrà caricato comparirà un segno di spunta alla sua estrema sinistra. Cliccando su questo segno di spunta apriremo una finestra di dialogo tramite la quale potremo fornire dei dati in input al gadget. Inoltre avvisiamo che tale finestra sarà definita all'interno del file settings.html

2) Con l'istruzione System.Gadget.onSettingsClosed=SettingsClosed; facciamo in modo che ogni volta che l'utente chiuderà la finestra di dialogo delle opzioni sia eseguita la funzione SettingsClosed

3) Nella funzione SettingsClosed con le istruzioni che seguono

```
   function SettingsClosed() {
colore=System.Gadget.Settings.read("colore");
            setContentText();
        }
```

Leggiamo il valore di una variabile associata al gadget e infine ridisegniamo la finestra con la funzione setContentText che avevamo precedentemente definito. Ok, fin qui non è troppo complicato. Dobbiamo ora gestire la finestra di dialogo delle opzioni. Per farlo dovremo creare un file settings.html. Il suo contenuto è il seguente:

```
<html>
    <head>
        <title>ioP Settings</title>
        <style>
            body {
                width:250;
                height:75;
            }
        </style>
     <script type="text/javascript"
                src="scripts/options.js"></script>
    </head>
    <body onload="init();">
        <span id="variabile" style="font-family:
                    Tahoma;font-size:10pt;">Colore

        <select id="envVar">
            <option value
                        ="white">Bianco</option>
            <option value ="green">Verde</option>
            <option value ="black">Nero</option>
            <option value ="red">Rosso</option>
         </select>
              </span>
    </body>
</html>
```

In questo file includiamo il sorgente options.js di cui parleremo fra un minuto e definiamo l'interfaccia della finestra di dialogo, semplicemente aggiungendovi un combobox contenente un elenco di colori. Infine in relazione all'evento OnLoad della finestra di dialogo richiamiamo la funzione init() che appunto è contenuta in options.js.
Il nostro option.js è il seguente:

```
// JScript source code

System.Gadget.onSettingsClosing =
                                    settingsClosing;
function init() {
            var
    currentSetting=System.Gadget.Settings.read
                                    ("actualcol");

            if (currentSetting !="")
envVar.selectedIndex=currentSetting;
}
function settingsClosing(event) {
            if (event.closeAction ==
                        event.Action.commit) {
                colore = envVar.value;
                actualcol=envVar.selectedIndex;

System.Gadget.Settings.write("colore",colore);

System.Gadget.Settings.write
                        ("actualcol",actualcol);
            }
            event.cancel=false;
}
```

In relazione all'evento OnLoad viene richiamata la funzione init() come già detto. In questa funzione si controlla se c'è qualche variabile actualcol già definita, se c'è il combo delle opzioni viene settato alla variabile corrente.
La funzione probabilmente più interessante è la settingsClosing()che viene richiamata quando la finestra di dialogo viene chiusa grazie all'istruzione:

```
System.Gadget.onSettingsClosing =
                              0settingsClosing;
```

Nella settingsClosing si preleva il valore selezionato nella combo e lo si scrive fra le variabili globali che disponiamo per personalizzare il colore dei caratteri. Quando la finestra di dialogo viene chiusa, il controllo torna a ciaomondo.html che come abbiamo già visto recupera il valore del colore e ridisegna il contenuto della pagina con

una setContentText().
Sembra veramente complicatissimo ma in realtà si tratta veramente di poche righe di codice. Una volta entrati nel meccanismo e capito come vanno disposti i file, vi accorgerete che si tratta veramente di operazioni piuttosto semplici sia sul piano semantico sia sintattico.

DISTRIBUIRE UN GADGET

Il deployment è probabilmente la parte più semplice di tutta la faccenda. In realtà i file con estensione .gadget non sono altro che degli zip rinominati. Per cui creare un package per la distribuzione si riduce ai seguenti passi:

1) Settate Windows Explorer in modo che vengano mostrare le estensioni anche per i file conosciuti

2) Create una cartella compressa

3) Trascinate i file associati con il gadget nella cartella in questione

4) Cambiate l'estensione della cartella da .zip a .gadget ed il gioco è fatto

potete rendere disponibile il gadget con una delle forme conosciute: localmente, hard disk, email o qualunque altro tipo di applicazione

CONCLUSIONI

I gadget di Windows Vista non rappresentano certamente una grande novità per il mondo dell'informatica. Chi usa Linux o Mac da molto tempo è abituato a meccanismi simili. Tuttavia si tratta di una di quelle comodità piccole ma sufficientemente utili da garantire spazi di mercato enormi. In realtà avere un gadget presente sul desktop che segnala per esempio variazioni di mercato, disponibilità di magazzino, quotazioni di borsa o altre notizie utili per un'operatore, può rappresentare una soluzione non di poco conto. Tutto viene dunque demandando alla capacità diu un buon programmatore di integrare nelle proprie applicazioni i vari gadget. Non si tratta neanche di una tecnologia invasiva, perciò non è neanche necessario modificare codice preesistente. Si tratta perciò di quelle tecniche che devono entrare a far parte del bagaglio tecnico di ogni buon programmatore Windows.

USARE AJAX NEI DISPOSITIVI MOBILI

L'ULTIMA MODA NELL'AMBITO DELLO SVILUPPO DI APPLICAZIONI WEB È SICURAMENTE AJAX. IMPARIAMO AD UTILIZZARE QUESTA TECNOLOGIA INNOVATIVA ANCHE NEI BROWSER DEI CELLULARI, CHE TIPICAMENTE IMPONGONO QUALCHE RESTRIZIONE...

Negli ultimi mesi si è molto parlato di argomenti quali le "chiamate asincrone" oppure "AJAX" o anche "ATLAS". Tutti questi argomenti afferiscono alla possibilità di migliorare enormemente l'esperienza dell'utente (user experience), dato che li loro utilizzo consente l'aggiornamento di pagine web in modo parziale, evitando il vecchio modello di "post-back" dove la pagina viene aggiornata nella sua totalità.
Queste nuove tecniche di aggiornamento si basano su Javascript e sul modello ad oggetti del navigatore (Document Object Model - DOM), consentono di effettuare chiamate asincrone ad una risorsa sul server web e conseguentemente processare le relativa risposta.
Sul versante desktop si è già potuto notare il fiorire di numerosi framework per lo sviluppo di soluzioni "AJAX-based"; tra tutti possiamo citare la recente uscita del framework di Microsoft, ASP.NET AJAX (già noto come ATLAS) che mette a disposizione degli sviluppatori tutta una serie di strumenti lato server che consentono lo sviluppo di applicazioni complete, senza (quasi) dover scrivere una linea di codice. A tutt'oggi questi potenti strumenti si appoggiano sulle possibilità offerte dai navigatori installati su macchine desktop. Purtroppo il discorso cambia notevolmente se parliamo di navigatori installati su dispositivi mobili. Per ovvie ragioni, questi ultimi supportano un modello ad oggetti (Document Object Model - DOM) più "povero" rispetto alle rispettive versioni desktop; per tale ragione le soluzioni implementate, utilizzando gli attuali framework di sviluppo, non rendono usufruibili le applicazioni dai dispositivi mobili.
In realtà è proprio nell'ambito dei dispositivi mobili che bisognerebbe concentrare lo sviluppo di applicazioni che ottimizzano il flusso dei dati via etere; sia per ragioni di "banda" dato che, in attesa del HSDPA, l'attuale copertura UMTS raramente consente una velocità apprezzabile e la velocità media consentita dalla rete GPRS è paragonabile a quella consentita dai vecchi modem analogici che utilizzavamo in casa prima dell'avvento dell'ADSL. Un ulteriore aspetto da tenere in considerazione è quello economico dato che il costo delle connessioni mobili, in genere, viene calcolato sul traffico; quindi minori saranno i dati che scambiamo con il server, minore sarà il peso della bolletta.
Premesso quanto sopra, sarà sicuramente interessante leggere il seguente articolo, dove vedremo come utilizzare AJAX per creare applicazioni web usufruibili anche dai dispositivi mobili.

LE FONDAMENTA DI AJAX

Alla base di tutto ciò che possiamo sviluppare in AJAX, c'è un particolare oggetto implementato nel modello ad oggetti (Document Object Model - DOM) del navigatore: XmlHttpRequest. Questo oggetto, in verità, non è una novità tecnologica degli ultimi tempi, dato che fu implementato la prima volta dalla Microsoft che lo introdusse in Internet Explorer 5. Il gruppo di sviluppo di Mozilla (su cui si basa il navigatore FireFox) e Opera Software (produttore dell'omonimo navigatore) seguirono a breve.
In ambito mobile, la disponibilità di navigatori è sicuramente maggiore rispetto all'ambito desktop. Sicuramente ci sono alcuni prodotti che hanno il vantaggio di essere gratuiti ed installati nelle ROM dei dispositivi (Internet Explorer Mobile su piattaforma Microsoft, piuttosto che Blazer nei dispositivi Treo), ma altri prodotti si battono tenacemente in un mercato che non è così monopolizzato come nell'ambito desktop (possiamo citare ad esempio Opera Mobile piuttosto che NetFront).
Qualsiasi sia il navigatore installato sul nostro dispositivo, affinché questo possa usufruire di applicazioni basate su AJAX, questo deve supportare l'oggetto XmlHttpRequest. Di seguito vediamo quanto sia semplice verificare questa caratteristica.

L'OGGETTO XMLHTTPREQUEST

Per verificare se il nostro navigatore preferito supporta AJAX, è sufficiente provare a creare, in una funzione Javascript, una istanza dell'oggetto XmlHttpRequest. Detta in questi termini la cosa sembra

alquanto semplice da verificare; in realtà dobbiamo prima capire "in che modo" il navigatore espone (eventualmente) l'oggetto.
Se il navigatore supporta in modo nativo l'oggetto, l'istruzione da testare sarà:

```
request = new XMLHttpRequest();
```

In ambiente Windows Mobile la situazione è leggermente differente dato che Pocket Internet Explorer (o Internet Explorer Mobile, che dir si voglia) supporta XmlHttpRequest come oggetto ActiveX; non solo, nel passaggio dalla versione Windows Mobile 2003 Second Edition alla versione Windows Mobile 5 del sistema operativo, il navigatore ha avuto delle evoluzioni, specialmente nel modello ad oggetti; anche il supporto all'oggetto XmlHttpRequest ha subito un cambiamento, quindi se vogliamo creare una istanza in ambiente Windows Mobile 2003 Second Edition dovremo scrivere la seguente istruzione:

```
request = new ActiveXObject("Microsoft.XMLHTTP");
```

In ambiente Windows Mobile 5 invece l'istruzione sarà:

```
request = new ActiveXObject("Msxml2.XMLHTTP");
```

Se mettiamo assieme le varie opzioni, possiamo creare una semplice applicazione "cross-browser" in grado di verificare su qualsiasi navigatore installato sul nostro dispositivo l'eventuale supporto dell'oggetto XmlHttpRequest.

```
var request;
function CreateRequest()
 {
    request = false;
    try {
      // Navigatore con supporto nativo
      request = new XMLHttpRequest();
      alert("Questo Navigatore supporta XMLHttp
                            Request in modo nativo.");
    } catch (PIE5) {
      try {
        // Internet Explorer su Windows Mobile 5
        request = new ActiveXObject("Msxml2.
                                        XMLHTTP");
        alert("Questo Navigatore supporta XMLHttp
                    Request tramite Msxml2.XMLHTTP");
      } catch (PIE2003) {
        try {
          // Internet Explorer su Windows Mobile 2003
                                     Second Edition
          request = new ActiveXObject
                              ("Microsoft.XMLHTTP");
          alert("Questo Navigatore supporta XMLHttp
                 Request tramite Microsoft.XMLHTTP");
        } catch (error) {
          request = false;
        }
      }
    }
    if (!request)
      alert("Questo Navigatore non supporta XMLHttp
                                         Request!");
    return request;
 }
```

Se incapsuliamo la precedente funzione in una pagina HTML in cui poniamo un solo bottone che esegue la suddetta funzione, abbiamo creato un semplice tool per verificare se il nostro navigatore preferito supporta l'oggetto XmlHttpRequest.

```
<body>
  <form id="form1" runat="server">
    <table align="center">
    <tr><td colspan="2" align="center">
    <h2>Test AJAX</h2>
    </td></tr>
    <tr><td align="center">
    <input type="Button" Value="Test AJAX"
          OnClick="Javascript:CreateRequest();" />
    </td>
    <td align="left"><img src="SmartAjax.gif"
                                          /></td>
    </tr>
    </table>
  </form>
</body>
```

Premesso che per mostrare i risultati del lavoro svolto nel presente articolo verrà utilizzato il prodotto Microsoft Device Emulator, nella figura seguente possiamo vedere il test effettuato rispettivamente su Pocket Internet Explorer su piattaforma Windows Mobile 2003 Second Edition per Smartphone e su Internet Explorer Mobile su piat-

Fig. 1: Il navigatore incorporato nei dispositivi basati sulle ultime versioni di Windows Mobile (2003SE e 5) supporta AJAX.

MICROSOFT DEVICE EMULATOR

Microsoft Device Emulator è compreso in Microsoft Visual Studio 2005 oppure è possibile scaricarlo dal sito Microsoft a partire dall'indirizzo http://msdn.microsoft.com/mobility/windowsmobile/downloads/default.aspx dove è possibile scaricare anche l'SDK di Windows Mobile 5.0 per Pocket PC e per Smartphone; questo componente aggiunge gli emulatori per la piattaforma Windows Mobile 5.0 al Device Emulator.

SUL WEB

Il navigatore Opera Mobile è scaricabile, nelle sue innumerevoli versioni, a partire dall'indirizzo: http://www.opera.com/products/mobile/products/. **Il navigatore NetFront è scaricabile, nelle sue innumerevoli versioni a partire dall'indirizzo:** http://www.access-company.com/products/netfrontmobile/browser/34_wm_tp.html.

taforma Windows Mobile 5 per Pocket PC (È possibile reperire la pagina testAJAX.htm nei file di supporto all'articolo).
In Figura 2 invece vediamo come navigatori di terze parti supportino l'oggetto XmlHttpRequest in modo nativo; il test è stato effettuato utilizzando rispettivamente Opera Mobile 8.6u2 e NetFront 3.4

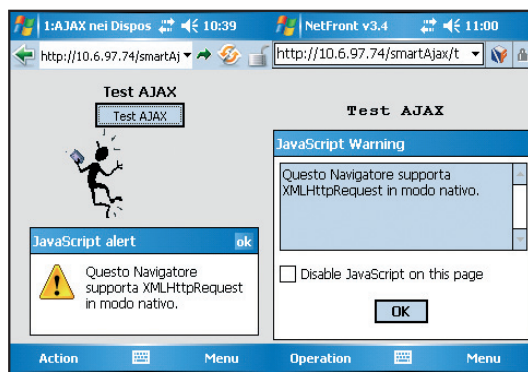



Fig. 2: lOpera Mobile e NetFront supportano AJAX in modo nativo.

CONFIGURAZIONE ED INVIO DELLA RICHIESTA

Dopo aver verificato che il nostro browser supporta l'oggetto XmlHttpRequest, possiamo compiere un passo in avanti e realizzare un primo colloquio tra il navigatore ed una risorsa sul web. Fortunatamente l'oggetto XmlHttpRequest è relativamente semplice e ci consente di costruire una infrastruttura basata su AJAX con pochi metodi e proprietà.
Al fine di costruire una infrastruttura modulare e, quindi, riusabile in diverse pagine, costruiamo una funzione Javascript in cui supponiamo che l'indirizzo della risorsa web con cui interloquire venga passata come parametro.
All'interno della funzione Javascript, creiamo innanzitutto l'oggetto XmlHttpRequest utilizzando una funzione analoga a quella vista in precedenza:

```
xmlHttpReq = CreateRequest();
```

Dobbiamo altresì configurare la richiesta; lo possiamo fare utilizzando il metodo open() dell'oggetto XmlHttpRequest, a cui passiamo, come primo parametro, il tipo di richiesta (che può essere "GET" o "POST"); come secondo parametro, l'indirizzo della risorsa web a cui connettersi; come terzo parametro un valore booleano che, se valorizzato a true, indica che la richiesta sarà asincrona.

```
xmlHttpReq.open("GET", url, true);
```

Nel caso in cui la risorsa a cui ci connettiamo necessiti di autenticazione, possiamo utilizzare la versione a 5 parametri del metodo open(); potremo specificare un nome utente nel quarto parametro e la relativa password nel quinto parametro.
Prima di inviare la richiesta al server dobbiamo ancora definire il meccanismo di "call-back" in modo che il server sia in grado di inviare le informazioni richieste, al navigatore.
Per ora ci occupiamo di agganciare alla richiesta una funzione, che verrà invocata una volta che i dati saranno disponibili. Dobbiamo valorizzare la proprietà onreadystatechange dell'oggetto XmlHttpRequest che rappresenta il gestore dell'evento associato alla variazione dello stato della richiesta; valorizziamo la proprietà con il nome della funzione che vogliamo utilizzare come gestore della risposta del server (la funzione è descritta nella prossima sezione).

```
xmlHttpReq.onreadystatechange = ManageResponse;
```

Come ultima operazione dobbiamo inviare la richiesta al server; utilizziamo il metodo send() dell'oggetto XmlHttpRequest a cui si può passare opzionalmente un parametro per inviare contenuto al server:

```
xmlHttpReq.send(null);
```

È possibile altresì inviare parametri sulla linea di comando, compilando opportunamente l'URL della richiesta.
Di seguito la funzione SendRequest().

```
function SendRequest(url)
{
    // Creiamo l'oggetto XmlHttpRequest
    xmlHttpReq = CreateRequest();
    // Configuriamo la richiesta
    xmlHttpReq.open("GET", url, true);
    // Impostiamo la funzione di call-back
    xmlHttpReq.onreadystatechange =
                                        ManageResponse
    // Inviamo la richiesta al server
    xmlHttpReq.send(null);
}
```

Gestione della risposta del server
Per gestire la risposta del server alla nostra richiesta, dobbiamo implementare la funzione che viene invocata ogni volta che varia lo stato della richiesta.
Lo stato della richiesta può assumere valori che vanno da 0 (uninitialized) a 4 (complete) secondo la tabella seguente:

Stato	Descrizione
0	uninitialized
1	loading
2	loaded
3	interactive
4	Complete

Chiaramente l'elaborazione dei dati deve avvenire solo dopo che la richiesta ha raggiunto lo stato "complete".

```
if (xmlHttpReq.readyState == 4)
```

Prima di procedere nell'analisi della risposta, è necessario verificare che l'interazione non abbia generato errori. Attraverso le proprietà status e statusText possiamo verificare il codice di ritorno della richiesta HTTP. Il codice 200 (il cui relativo valore testuale è "OK") ci segnala che il colloquio non ha generato errori quindi possiamo elaborare la risposta.

```
if (xmlHttpReq.status == 200)
```

In caso di errore lo stato verrà valorizzato con i valori che a volte riscontriamo durante la navigazione: 500 ("Internal Server Error"), 404 ("Not Found"), ecc.
L'oggetto XmlHttpRequest mette a disposizione due proprietà attraverso cui possiamo elaborare la risposta del server: responseText contiene la risposta del server in formato stringa mentre responseXML contiene la risposta del server in formato XML, in particolare, se la risorsa invocata è un web service, il documento XML ritornato, rappresenta il messaggio SOAP inviato dal server.

```
UpdatePage(xmlHttpReq.responseText);
```

Nell'esempio corrente la nostra funzione passerà semplicemente il contenuto della risposta ad una funzione che dovrà essere implementata al livello della pagina chiamante.

```
function ManageResponse()
{
   // Verifichiamo che la richiesta sia completata
   if (xmlHttpReq.readyState == 4)
   {
      // Verifichiamo che il colloquio non abbia generato
                                                  errori
      if (xmlHttpReq.status == 200)
      {
         // Passiamo la risposta del server ad una
      funzione che gestirà il risultato al livello della pagina
      UpdatePage(xmlHttpReq.responseText);
      }
   }
}
```

AJAX ENGINE

A questo punto della trattazione possiamo effettuare un piccolo riassunto per constatare che, così come abbiamo strutturato le funzioni, abbiamo creato in realtà un "motore" AJAX modulare e riusabile. In input al nostro motore va l'indirizzo (completo di eventuali parametri) della risorsa web che andremo ad interrogare; l'output della richiesta verrà passato ad una funzione (UpdatePage) che dovrà essere implementata in ogni pagina che utilizzerà il nostro motore.
In teoria, proprio per rendere il nostro motore riusabile, potremmo porre le tre funzioni all'interno di un file (ad esempio AJAXEngine.js) che potrebbe essere semplicemente linkato in ogni pagina con la seguente istruzione:

```
<script language="javascript" src="AJAXEngine.js">
                                          </script>
```

Purtroppo i navigatori mobili sembrano non "intendere" questa direttiva quindi dovremo copiare il codice sviluppato finora in tutte le pagine che intendiamo rendere "AJAX-based".

L'AGGIORNAMENTO PARZIALE

Finora abbiamo costruito una infrastruttura che gestisce in tutto e per tutto la comunicazione asincrona tra il nostro navigatore ed una qualsivoglia risorsa web; non abbiamo dedicato nessuna parola all'aggiornamento dinamico delle pagine, alla possibilità di aggiornare solo parti e non l'itera pagina.
Con XmlHttpRequest gestiamo la comunicazione asincrona tra client (il nostro navigatore, ma non solo) ed una risorsa web. L'aggiornamento o meglio l'elaborazione della risposta del server è compito del codice lato client che, utilizzando gli oggetti messi a disposizione del DOM (Document Object Model), "inietta" i dati nella pagina, andando ad aggiornare le parti rilevanti della stessa.
È proprio questa la ragione che limita (almeno ad oggi) l'utilizzo dei framework di sviluppo di applicazioni "AJAX-based" sui dispositivi mobili, ovvero l'utilizzo di oggetti che sono messi a disposizione dai DOM dei navigatori utilizzati nelle postazioni desktop, ma che purtroppo non trovano corrispondenza nei DOM messi a disposizione dai navigatori in ambito mobile.
Avere ben chiare quali siano le dotazioni "minime" messe a disposizione dagli strumenti "target" della nostra applicazione è fondamentale per ottenere la maggior fruibilità possibile.
Ad esempio è importante sapere che le proprietà innerText ed innerHTML sono supportate in ambiente Windows Mobile 2003 Second Edition SOLO dagli elementi <div> e <span> mentre sono supportate in tutti gli elementi su piattaforma Windows Mobile 5. La speranza è ovviamente affidata alla evoluzione dei framework di svi-

luppo auspicando il supporto anche dei dispositivi mobili nelle prossime versioni.

UN PRIMO ESEMPIO

Per chiudere il cerchio, ovvero per testare la nostra prima interazione AJAX, dobbiamo creare una risorsa web che, se interrogata, ci restituisca dei dati.
Come primo esempio utilizzeremo una semplice pagina aspx che restituisce la data di sistema (un esempio più complesso verrà trattato più avanti nella trattazione).
La pagina da caricare è FirstReq.htm ed è composta da un bottone che invoca la risorsa tramite AJAX, ed un elemento <div> in cui andremo a stampare la data.

```
<body>
  <form id="form1" runat="server">
    <table align="center">
    <tr><td colspan="2" align="center">
    <h2>Primo Esempio AJAX</h2>
    </td></tr>
    <tr><td align="center">
    <input type="Button" Value="Scarica Data"
                OnClick="Javascript:GetDate();" />
    </td>
    <td align="left"><img src="SmartAjax.gif" />
                                            </td>
    </tr>
    <tr><td colspan="2" align="center">
    La data di sistema è:<br />
    <div id="divSysDate" align="center"></div>
    </td></tr>
    </table>
  </form>
</body>
```

Per quanto riguarda il codice Javascript, oltre al motore AJAX, definiamo una funzione GetDate() che semplicemente incapsula la richiesta al server, e la funzione "obbligatoria" UpdatePage() per stampare la data del server nel suddetto elemento <div> della pagina.

```
<script language="javascript">
function GetDate()
{
    SendRequest('FirstReq.aspx');
}
function UpdatePage(result)
{
    divSysDate.innerHTML = result;
}
</script>
```

NOTA

DOVE TROVO L'EMULATORE!

Microsoft Device Emulator è compreso in Microsoft Visual Studio 2005 oppure è possibile scaricarlo dal sito Microsoft a partire dall'indirizzo http://msdn.microsoft.com/mobility/windowsmobile/downloads/default.aspx dove è possibile scaricare anche l'SDK di Windows Mobile 5.0 per Pocket PC e per Smartphone; questo componente aggiunge gli emulatori per la piattaforma Windows Mobile 5.0 al Device Emulator.

Nella funzione GetDate() possiamo notare che l'indirizzo verso cui effettuiamo la richiesta (FirstReq.aspx), è definito senza specificare per intero il protocollo ed il dominio (http://www.ilTuoDominio.it/ FirstReq.aspx); questa cosa non è frutto di una dimenticanza ma è frutto del modello di sicurezza di AJAX (cosiddetto a "SandBox") che consente di inviare le richieste SOLO al dominio in è eseguito. Ciò significa che se invochiamo la pagina FirstReq.htm dal nostro desktop, potremo invocare solo risorse presenti nel nostro desktop; analogamente, se invochiamo una pagina memorizzata in http://www.ilTuoDominio.it, potremo invocare solo risorse presenti in http://www.ilTuoDominio.it. Lato server, la pagina che fungerà da risorsa web è FirstReq.aspx. Nel codice eseguito al caricamento della pagina (Page_Load()), dobbiamo semplicemente stampare in output il valore della data. Quindi dobbiamo terminare l'esecuzione della pagina dato che tutti i dati necessari sono stati inviati.

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write(DateTime.Now.ToString());
    Response.End();
}
```

Nella figura seguente andiamo a verificare il funzionamento della nostra infrastruttura utilizzando Pocket Internet Explorer, Opera Mobile e NetFront, tutti su piattaforma Windows Mobile 5.
Chiaramente dalla figura non è possibile constatare come la pagina venga aggiornata SENZA ricaricare né gli elementi grafici né il bottone.

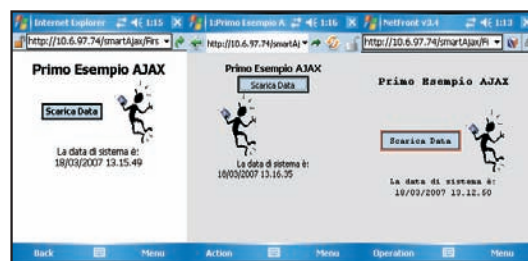



Fig. 3: La nostra prima interazione AJAX.

ATTENZIONE AI FILE TEMPORANEI!!!

Prima di proseguire con un esempio più complesso, è il caso di osservare innanzitutto coma la stessa pagina venga mostrata in modo "abbastanza" diverso nei tre navigatori (addirittura Opera Mobile manda alla riga successiva l'immagine). Questo ci serva solo come "appunto" da tenere a mente per tutte le successive applicazioni che svilupperemo: l'uniformità di presentazione delle pagi-

ne nei dispositivi mobili è realmente ancora una chimera. Una seconda cosa da evidenziare in questo esempio, è il comportamento assolutamente diverso dei tre navigatori nei confronti dei dati memorizzati nei file temporanei.
Se noi proviamo ad eseguire il codice così come definito nella sezione precedente, avremo i seguenti comportamenti:

- Internet Explorer: Dopo aver caricato la prima volta la data NON c'è verso di aggiornarla; premendo ripetutamente il tasto "Scarica Data", il valore resta inalterato. Anche se proviamo ad eliminare i file temporanei, il risultato non cambia. Per avere un nuovo valore è necessario TERMINARE il programma (non premendo la "X" in alto a destra ma andando in Impostazioni->Sistema->Memoria->Programmi in esecuzione per terminare Internet Explorer).
- Opera Mobile: Dopo aver caricato la prima volta la data, il valore resta inalterato finché non eliminiamo i file temporanei (menu->Tools->Settings->History->clear cache).
- NetFront: Il programma si comporta correttamente senza alcuna modifica.

Questi comportamenti sono dovuti ad una gestione "particolare" dei file temporanei in cui (evidentemente) i tre navigatori si differenziano.
Per ovviare a questo problema dobbiamo evitare che la pagina venga posta in "cache" ovvero non deve rimanere traccia nei file temporanei del dispositivo. Il codice da inserire, allora, nella risorsa web (ovvero nella pagina FirstReq.aspx) sarà:

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.CacheControl = "no-cache";
    Response.AddHeader("Pragma", "no-cache");
    Response.Expires = -1;
    Response.Write(DateTime.Now.ToString());
    Response.End();
}
```

Le prime tre righe effettivamente evitano che la pagina resti memorizzata, con questa modifica il codice funziona perfettamente in ogni navigatore.

UN ESEMPIO PIÙ COMPLESSO

Nel secondo esempio interrogheremo una base dati passando anche un parametro.
Fermo restando che il motore AJAX utilizzato è esattamente lo stesso dell'esempio precedente, concentriamoci sulla pagina SecondReq.htm il cui corpo è riportato di seguito:

```
<body>
  <form id="form1" runat="server">
    <table align="center">
    <tr><td colspan="2" align="center">
    <h2>AJAX nei Dispositivi Mobili</h2>
    </td></tr>
    <tr><td align="center">
    <select id="DropDownList1">
      <option value="M">Maschi</option>
      <option value="F">Femmine</option>
    </select>
    <input type="Button" Value="Cerca Impiegati"
       OnClick="Javascript:GetEmployees(Drop
                                   DownList1);" />
    </td>
    <td align="left"><img src="SmartAjax.gif" />
                                           </td>
    </tr>
    <tr><td colspan="2">
    <div id="divEmployees"></div>
    </td></tr>
    </table>
  </form>
</body>
```

L'obiettivo è quello di interrogare una tabella del database di esempi AdventureWorks incorporato in Microsoft Sql Server 2005. In particolare si vuol ottenere la lista dei dipendenti di sesso maschile o femminile a partire dalla selezione della relativa voce presente nella lista a cascata.
La pressione del bottone "Cerca Impiegati" invoca la funzione Javascript GetEmployees():

```
function GetEmployees(obj)
 {
    divEmployees.innerHTML = "Invio richiesta in
                                      corso ...";
    SendRequest('GetList.aspx?gender=' + obj.
                                          value);
 }
```

Come possiamo vedere dal codice precedente, prima di inviare la richiesta al server (SendRequest()) dobbiamo informare l'utente che una richiesta è in corso; questo perché, visto che la richiesta è asincrona, la pagina continua ad essere "attiva" durante l'interazione. In generale è buona norma informare l'utente che alcune operazioni sono in corso, in modo che la pagina apparentemente "inattiva" non lo disorienti. Sul server viene invocata la pagina aspx nominata GetList.aspx. Anche in questo caso tutta la logica applicativa è contenuta nel metodo di gestione del caricamento della pagina (Page_Load()).

```
public partial class GetList : System.Web.UI.Page
{
```

```
private string strConnString =
ConfigurationManager.ConnectionStrings["DB
ConnectionString"].ConnectionString;

protected void Page_Load(object sender, Event
Args e)
{
    // Reperiamo il parametro dalla stringa in input
    string strGender = Request.QueryString
["gender"];
    // Definiamo la connessione al DB
    SqlConnection oConn = new SqlConnection
(strConnString);
    // Definiamo il comando da inviare al DB
    SqlCommand oRetCommand = new Sql
Command();
    // Compiliamo la stringa che rappresenta
l'interrogazione
    string strCommandText = "SELECT ISNULL
(c.Title, '') + ' ' + ISNULL(c.FirstName, '') + ' ' + " +
        "ISNULL(c.LastName, '') AS Employee,
c.EmailAddress as EMail " +
        "FROM Person.Contact c " +
        "inner join HumanResources.Employee e ON
c.ContactId = e.ContactId " +
        "where e.Gender='" + strGender + "';";
    // Imponiamo il testo del comando e la
connessione
    oRetCommand.CommandText = strCommand
Text;
    oRetCommand.Connection = oConn;
    // Scarichiamo i dati su di un DataSet
    SqlDataAdapter oSqlDataAdapter = new Sql
DataAdapter(oRetCommand);
    DataSet ds = new DataSet();
    oSqlDataAdapter.Fill(ds, "dataSet");
    // Creiamo un oggetto di tipo GridView
    GridView gv = new GridView();
    // Valorizziamo la sorgente dati al DataSet
appena creato
    gv.DataSource = ds;
    // Effettuiamo il bind dei dati
    gv.DataBind();
    HtmlTextWriter htw = new HtmlTextWriter
(Response.Output);
    // Inviamo i dati al client
    gv.RenderControl(htw);
    Response.End();
  }
}
```

SQL SERVER 2005 ED ADVENTUREWORKS

Per eseguire il codice del secondo esempio, possiamo scaricare Microsoft Sql Server 2005 Express Edition dall'indirizzo: http://www.microsoft.com/downloads/details.aspx?familyid=220549b5-0b07-4448-8848-dcc397514b41&displaylang=en **mentre il database di esempi AdventureWorks è scaricabile dall'indirizzo:** http://www.microsoft.com/downloads/details.aspx?familyid=9697AAAA-AD4B-416E-87A4-A8B154F92787&displaylang=en. **Ricordiamoci di definire una login in Sql Server coerentemente con quella definita nel file web.config per accedere ai dati e che troviamo nei file di supporto all'articolo. Altrimenti ricordiamoci di cambiare la stringa di connessione in modo da poter accedere ai dati.**

```
<connectionStrings>
  <add name="DBConnectionString"
      connectionString="Data Source=(local);Initial Catalog=AdventureWorks;
      User ID=adWorksUser;Password=passguor"
   providerName="System.Data.SqlClient" />
</connectionStrings>
```

Come possiamo notare dal codice precedente, come prima cosa ricaviamo il parametro dalla linea di comando (per semplicità non vengono fatti controlli per la validazione dell'input; in un ambiente di produzione è assolutamente necessario validare l'input a maggior ragione se la nostra applicazione è esposta su Internet). Di seguito si stabilisce una connessione al database, quindi si esegue la richiesta, il cui risultato è immagazzinato in un oggetto di tipo DataSet (anche in questo caso, per semplicità viene compilata la richiesta in modo testuale nel codice; in ambiente di produzione è buona norma usare le stored procedure). A questo punto i dati che ci servono sarebbero pronti per essere inviati alla pagina che li ha richiesti. In verità, in questo esempio, vediamo come sia possibile farci "aiutare" dal server non solo nel reperimento dei dati, ma anche nella loro formattazione. Creiamo un oggetto di tipo GridView e lo accoppiamo con il DataSet appena creato

```
// Creiamo un oggetto di tipo GridView
GridView gv = new GridView();
// Valorizziamo la sorgente dati al DataSet appena
creato
gv.DataSource = ds;
// Effettuiamo il bind dei dati
gv.DataBind();
```

Creiamo un oggetto HtmlTextWriter e definiamo come destinazione dei dati, proprio il flusso in uscita ovvero il flusso che ricondurrà i dati alla pagina che ne ha fatto richiesta.

```
HtmlTextWriter htw = new HtmlTextWriter(Response.
Output);
```

Fig. 4: Un esempio più complesso.

Infine utilizziamo il metodo RenderControl dell'oggetto GridView che non fa altro che riversare nel flusso di uscita il codice HTML che rappresenta la formattazione grafica dell'oggetto GridView, ovvero una struttura <table> (una tabella).

```
// Inviamo i dati al client
gv.RenderControl(htw);
```

Nell'immagine seguente possiamo vedere i vari passi dell'interazione tratti dai tre navigatori che abbiamo scelto di utilizzare. Apriamo la pagina con Opera Mobile, lanciamo l'interazione con NetFront, vediamo il risultato su Internet Explorer.
Come ultima annotazione, possiamo notare che in questo caso non si presenta il problema rilevato nell'esempio precedente e relativo alla persistenza della pagina nei file temporanei (con relativo mancato aggiornamento).
Probabilmente, visto che la pagina varia "abbastanza", i navigatori si accorgono della variazione ed effettuano correttamente l'aggiornamento. Ovviamente mancando una certezza assoluta è consigliabile provare il nostro lavoro su più piattaforme e su più navigatori (almeno quelli che supponiamo vengano utilizzati dai nostri utenti).

ALCUNE CONSIDERAZIONI

Il codice dell'esempio appena visto ci porta a fare alcune considerazioni: sicuramente, l'aver utilizzato i "tool" messi a disposizione dal server ed, in questo caso, da ASP.NET 2.0 (l'oggetto GridView è appunto stato introdotto nella versione 2.0 del framework), ci ha consentito di "lavorare" molto meno lato client dato che i dati ci arrivano già formattati e quel che dobbiamo fare non è nient'altro che valorizzare un elemento <div> con ciò che ci proviene dal server.
D'altro canto però in questo modo abbiamo violato il principio secondo cui è buona norma sviluppare codice su livelli indipendenti, disaccoppiando l'accesso ai dati dalla logica applicativa e, a maggior ragione, dall'interfaccia utente. È pur vero che formattare una tabella da codice client è un "lavoro" che per il processore di un dispositivo mobile (ricordiamoci che questo codice può lavorare anche su di uno smartphone) potrebbe non essere indifferente, soprattutto se la tabella ha dimensioni notevoli. Indubbiamente è vero allo stesso tempo che inviare i dati già formattati in una struttura HTML comporta un sovraccarico della banda che oltre a trasportare i dati veicola anche i tag HTML (<table>, <tr>, <td>, ecc.); ovviamente resta da stabilire in che modo potremmo serializzare il DataSet per inviarlo alla nostra pagina. Tutte queste osservazioni servono solo a metterci in guardia rispetto al modo in cui andiamo a sviluppare le nostre applicazioni; dobbiamo cercare il giusto compromesso tra la gestibilità del codice, la sua scalabilità, la sua efficienza (sia in termini di tempo di esecuzione che in termini di dati scambiati che influenzano il costo economico della soluzione), ecc.; in questo articolo si è scelto di demandare la formattazione dei dati al server, violando il disaccoppiamento dei livelli ma ottenendo molto meno codice da gestire e, sicuramente una maggiore velocità di esecuzione.

CONCLUSIONI

L'ultima moda nell'ambito dello sviluppo di applicazioni web è sicuramente AJAX. Utilizzando diverse tecnologie come Javascript, il modello ad oggetti dei navigatori (Document Object Model - DOM) e l'XML, è oggi possibile ammirare numerosi siti (Google in testa a tutti) dove l'aggiornamento delle pagine non avviene più seguendo il classico paradigma del "post-back", dove l'intera pagina doveva essere ricaricata per aggiornare anche solo una parte di essa, ma avviene in maniera parziale e solo le porzioni da aggiornare subiscono effettivamente un cambiamento. Questo salto di qualità nella fruizione delle risorse web è coadiuvato ancor più dalla nascita di numerosi strumenti un grado di semplificare notevolmente lo sviluppo di siti basati su AJAX. Purtroppo questi strumenti si basano fortemente sulle caratteristiche avanzate dei navigatori disponibili in ambiente desktop, caratteristiche che non trovano corrispondenza nei "fratelli minori" disponibili nei dispositivi mobili. Nel presente articolo si è mostrato come in realtà sia possibile usufruire dei vantaggi offerti dalle soluzioni basate su AJAX anche in ambiente mobile. Il requisito minimo del nostro navigatore mobile preferito è il supporto dell'oggetto XmlHttpRequest che è alla base del colloquio asincrono tra il nostro dispositivo e le risorse web. Di seguito abbiamo costruito un piccolo motore AJAX, costituito da tre funzioni scritte in Javascript, necessarie per gestire il colloquio tra una risorsa server ed il nostro navigatore. Il relativo aggiornamento delle informazioni è stato demandato ad una funzione, scritta a livello della pagina chiamante, che deve gestire i limiti talvolta imposti da navigatori mobili.
Nell'ultima parte dell'articolo è stato illustrato un esempio in cui si è evidenziato il compromesso a cui è necessario sottostare quando si sviluppano soluzioni per dispositivi mobili che hanno inevitabilmente limiti e vincoli che in ambiente desktop non abbiamo.

Oscar Peli

L'AUTORE

Oscar Peli è .NET Solution Architect ed amministratore dei dispositivi mobili presso il Comune di Ancona. Come consulente ha sviluppato presso l'Università degli Studi di Macerata un sistema di pubblicazione di contenuti per il supporto a progetti di ricerca http://reti.unimc.it. Oscar è contattabile all'indirizzo: opeli@unimc.it.

MA QUANTE VISITE HA IL MIO SITO?

IMPARIAMO A SFRUTTARE IL FORMATO DEI LOG DI IIS IMPORTANDO I DATI IN UNA NOSTRA APPLICAZIONE E REALIZZANDO UN COMPLETO SISTEMA DI GESTIONE DELLE STATISTICHE. PER FARLO SFRUTTEREMO ANCHE SSIS...

CD WEB
IIS_Logs.zip
cdrom.ioprogrammo.it

Abbiamo un sito web e vogliamo monitorarne gli accessi senza dover acquistare costosi strumenti per le statistiche o affidarci a servizi esterni.
Perché non scriviamo noi un software per monitorare il traffico del nostro sito?
Utilizzando i log di IIS, Microsoft.Net, e SQL Server 2005 realizzeremo un semplice sistema di statistiche degli accessi.
Requisito per l'utilizzo del software che andremo a sviluppare è quello di avere l'accesso ai file di log tramite un percorso di rete oppure tramite ftp.

Figura 1: IIS Logs in azione!

I LOG DI IIS

IIS è dotato di un completo sistema per il log degli accessi. È possibile gestirlo tramite Internet Information Services Manager. Per aprire la maschera di configurazione, cliccare con il tasto destro del mouse sul nome del server e selezionare "Proprietà"; in basso spuntiamo il flag "Consenti registrazione attività". È poi possibile scegliere il formato di log tramite la tendina "Formato registro attivo" e quali informazioni loggare cliccando su "Proprietà".
Le informazioni che IIS riesce a loggare sono molte e sono suddivise in quattro categorie in base ad un prefisso:

- **s-** : azioni server
- **c-** : azioni client
- **cs-** : azioni client-to-server
- **sc-** : azioni server-to-client

Ecco, infine, alcune delle informazioni che è possibile ottenere:

Figura 2: Configuriamo il tipo di log e le informazioni da salvare

- **date**: la data dell'evento
- **time**: l'ora
- **c-ip**: l'ip del client
- **cs-method**: l'azione del client (GET, POST etc…)
- **cs-uri-stem**: la risorsa richiesta
- **cs-uri-query**: la querystring passata al server
- **sc-status**: lo stato HTTP dell'azione (es: 200 OK)
- **cs(User-Agent)**: il browser del client
- **cs(Referer)**: il sito da cui si proviene

I dati che IIS loggherà sono configurabili tramite la maschera "Proprietà registrazione estese".

```
#Software: Microsoft Internet Information Services 5.0
#Version: 1.0
#Date: 2007-02-10 16:43:24
#Fields: date time c-ip cs-method cs-uri-stem cs-uri-query sc-status cs(User-Agent) cs(Ref
```

Figura 3: Il nostro file di log

Per il nostro software decidiamo di memorizzare le informazioni contenute nella lista sopra descritta. Nella sezione "Proprietà generali" possiamo, invece, scegliere dove salvare il nostro file di log.
Gli accessi al sito saranno quindi monitorati e salvati in dei file come quello in figura.

IL DATABASE

Innanzitutto, creiamo il nostro database che conterrà i log.
Abbiamo bisogno di tre tabelle:

- **log_IISLogs**: la principale, conterrà i log
- **sta_status_codes**: conterrà i codici di stato
- **ext_extensions**: per le estensioni dei file

Ecco gli script per la creazione:

```
-- La tabella con i codici di stato
CREATE TABLE [dbo].[sta_status_codes](
        [sta_status] [varchar](10) COLLATE
                Latin1_General_CI_AS NOT NULL,
        [sta_description] [varchar](1000) COLLATE
                Latin1_General_CI_AS NULL,
 CONSTRAINT [PK_sta_status_codes] PRIMARY KEY
                CLUSTERED
(
        [sta_status] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

-- La tabella dei log
CREATE TABLE [dbo].[log_IISLogs](
        [log_id] [int] IDENTITY(1,1) NOT NULL,
        [log_datetime] [datetime] NULL,
        [log_client_ip] [varchar](15) COLLATE
                Latin1_General_CI_AS NULL,
        [log_method] [varchar](10) COLLATE
                Latin1_General_CI_AS NULL,
        [log_uri] [varchar](1000) COLLATE
                Latin1_General_CI_AS NULL,
        [log_uri_query] [varchar](1000) COLLATE
                Latin1_General_CI_AS NULL,
        [log_sta_status] [varchar](10) COLLATE
                Latin1_General_CI_AS NULL,
        [log_user_agent] [varchar](1000) COLLATE
                Latin1_General_CI_AS NULL,
        [log_referer] [varchar](1000) COLLATE
                Latin1_General_CI_AS NULL,
 CONSTRAINT [PK_log_IISLogs] PRIMARY KEY
                CLUSTERED
(
        [log_id] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

```
GO

-- Aggiungiamo il constraint
ALTER TABLE [dbo].[log_IISLogs]  WITH CHECK ADD
        CONSTRAINT [FK_log_IISLogs_sta_status_codes]
                FOREIGN KEY([log_sta_status])
REFERENCES [dbo].[sta_status_codes] ([sta_status])
GO

-- La tabella dei tipi di file
CREATE TABLE [dbo].[ext_extensions](
        [ext_extension] [varchar](4) COLLATE
                Latin1_General_CI_AS NOT NULL,
        [ext_type] [varchar](3) COLLATE
                Latin1_General_CI_AS NOT NULL,
 CONSTRAINT [PK_ext_extensions] PRIMARY KEY
                CLUSTERED
(
        [ext_extension] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Figura 4: Il database dei log

L'ARCHITETTURA

Il nostro software sarà formato da tre componenti:

- un client ftp: per effettuare il download del file di log
- un package di SSIS: per impostare i dati in SQL Server
- una web application: per visualizzare i dati

Un package SSIS schedulato giornalmente si occuperà di lanciare il client FTP per recuperare il file di log e, una volta scaricato tale file, di elaborarlo ed importarlo nella tabella di SQL Server.
La web application avrà il compito di visualizzare i dati relativi ad accessi, pagine viste, referals ed altro.

NOTA

SYSTEM.NET

Il namespace System.Net fornisce una serie di classi per la gestione dei protocolli di comunicazione. Tra quelle più utilizzate ricordiamo HttpWebRequest e HttpWebResponse per le richieste HTTP, o le nuove FtpWebRequest e FtpWebResponse per l'FTP.
Ulteriori informazioni sono disponibili all'indirizzo:
http://msdn2.microsoft.com/it-it/library/system.net(VS.80).aspx

IL CLIENT FTP

Iniziamo con la scrittura del client FTP.
Per la realizzazione della classe per l'FTP utilizzeremo il namespace System.Net che contiene due classi FtpWebRequest e FtpWebResponse le quali consentono di effettuare richieste FTP e leggere le relative risposte. Apriamo Visual Studio e creiamo un nuovo progetto di tipo "Console" chiamato IISLogFTPDownloadVB; creiamo quindi la classe FTPClient.vb che gestirà l'FTP.
All'interno della classe dichiariamo delle proprietà private:

```
Private _Protocol As String ' Il protocollo usato
Private _Host As String ' L'host
Private _FilePath As String ' Il persorso del file
Private _User As String ' Nome utente
Private _Pass As String ' Password
Private _SSL As Boolean ' Se collegarsi con SSL
Private _Passive As Boolean ' Se usare la modalità passiva
Private _Conn As FtpWebRequest ' La richiesta FTP
Private _Resp As FtpWebResponse ' La risposta FTP
```

e quindi il costruttore:

```
Public Sub New(ByVal Protocol As String, ByVal Host
    As String, ByVal User As String, ByVal Pass As String,
    ByVal SSL As Boolean, ByVal Passive As Boolean)

    _Protocol = Protocol
    _Host = Host
    _FilePath = ""
    _User = User
    _Pass = Pass
    _SSL = SSL
    _Passive = Passive

End Sub
```

che valorizza alcune delle proprietà della classe.

NOTA

IL CODICE ALLEGATO

Gli esempi nell'articolo sono in Visual Basic; per chi fosse interessato, nel CD allegato è presente tutto il codice sorgente anche in C#.

IL METODO CONNECT

```
Private Sub Connect()
    _Conn = CType(WebRequest.Create(_Protocol &
        "://" & _Host & _FilePath), FtpWebRequest)
    _Conn.EnableSsl = _SSL
    _Conn.UsePassive = _Passive
    _Conn.Credentials = New
        NetworkCredential(_User, _Pass)
End Sub
```

crea una WebRequest alla risorsa, la "casta" come FtpWebRequest ed imposta i parametri di connessione.
Cuore della classe è, infine, la funzione Download:

```
Public Function Download(ByVal FilePath As String,
        ByVal FileDestination As String) As Integer

    Try

        _FilePath = FilePath

        ' Connessione all'host ftp
        Connect()

        ' Imposto il tipo di richiesta
        _Conn.Method =
            WebRequestMethods.Ftp.DownloadFile

        ' Ottengo la risposta e la inserisco in uno stream
        _Resp = CType(_Conn.GetResponse(),
            FtpWebResponse)
        Dim _RespStream As Stream =
            _Resp.GetResponseStream()

        ' Salvo lo stream in un file leggendolo a blocchi di byte
        Dim _File As FileStream = New
            FileStream(FileDestination, FileMode.Create,
            FileAccess.Write)

        Try
            Dim buf(1024) As Byte
            Dim bytesRead As Integer =
                _RespStream.Read(buf, 0, buf.Length)
            While (bytesRead <> 0)
                _File.Write(buf, 0, bytesRead)
                bytesRead = _RespStream.Read(buf, 0,
                    buf.Length)
            End While

            Return 0

        Finally

            _File.Close()
            _RespStream.Close()

        End Try

    Catch ex As WebException
        If (ex.Message.Contains("550")) Then
            Return 550 ' File non trovato
        ElseIf (ex.Message.Contains("530")) Then
            Return 530 'Accesso negato
        ElseIf (ex.Message.Contains(_Host)) Then
            'Host non trovato
            Return 551
        Else
            Return -2 ' Errore generico
        End If
    Catch e As DirectoryNotFoundException
```

```
            Return -500
        Catch
            Return -2   'Errore generico
        End Try

End Function
```

Vediamo come lavora:
legge, come prima cosa, il percorso del file da scaricare e lo salva nella proprietà privata _FilePath, quindi, lancia il metodo Connect() per collegarsi al server FTP.
Tramite la riga

```
_Conn.Method =
                    WebRequestMethods.Ftp.DownloadFile
```

viene impostato il metodo della WebRequest a "DownloadFile".
Dopo aver effettuato la connessione, leggiamo la risposta tramite il metodo GetResponse() della classe FtpWebRequest e da questa otteniamo uno Stream di dati utilizzando il metodo GetResponse Stream()

```
_Resp = CType(_Conn.GetResponse(),
                                    FtpWebResponse)
Dim _RespStream As Stream =
                         _Resp.GetResponseStream()
```

Ottenuto lo stream di dati, il più è fatto: è sufficiente salvarlo su file. Per farlo, utilizziamo un ciclo che legge i dati dallo stream a blocchi di byte e li salva in un FileStream.
La funzione restituisce un valore intero. Il valore restituito sarà zero se il download va a buon fine, altrimenti viene restituito un codice di errore in base al tipo di eccezione che si è verificata. La funzione GetErrorMessage converte il codice di errore in un messaggio "user friendly".
Realizzata la classe per l'FTP, occupiamoci del programma. Aggiungiamo un file di configurazione app.config al progetto. Vi memorizzeremo le informazioni per l'accesso FTP.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="Protocol" value="ftp"/>
    <add key="Host" value="ftp.nomeserver.it"/>
    <add key="BaseFilePath"
                 value="/cartellainiziale/cartellalogs/"/>
    <add key="User" value="username"/>
    <add key="Pass" value="password"/>
    <add key="SSL" value="false"/>
    <add key="Passive" value="true"/>
    <add key="FileDestination"
                        value="C:\IISLogs\IISLog.txt"/>
```

```
    <add key="ErrorsFile"
                         value="C:\IISLogs\ERR.txt"/>
  </appSettings>
</configuration>
```

Per utilizzare il download da percorso di rete, occorre cambiare due righe del web.config

```
<add key="Protocol" value="share"/>
<add key="BaseFilePath"
                 value="\\NomeShare\NomeCartella\"/>
```

Vediamo infine il Main del programma:

```
    Sub Main()

        'Variabili da configurazione
        Dim Protocol As String =
              ConfigurationManager.AppSettings("Protocol")
        Dim Host As String =
                  ConfigurationManager.AppSettings("Host")
        Dim BaseFilePath As String =
          ConfigurationManager.AppSettings("BaseFilePath")
        Dim User As String =
                  ConfigurationManager.AppSettings("User")
        Dim Pass As String =
                  ConfigurationManager.AppSettings("Pass")
        Dim SSL As String =
     Convert.ToBoolean(ConfigurationManager.AppSettings
                                                ("SSL"))
        Dim Passive As String =
     Convert.ToBoolean(ConfigurationManager.AppSettings
                                            ("Passive"))
        Dim FileDestination As String =
       ConfigurationManager.AppSettings("FileDestination")
        Dim ErrorsFile As String =
            ConfigurationManager.AppSettings("ErrorsFile")

        'Nome del file da scaricare
        Dim day As DateTime = DateTime.Now.AddDays(-
                                                     1)
        Dim filename As String = "ex" &
                       day.ToString("yyMMdd") & ".log"

        ' Scarico il file

        If Protocol = "ftp" Then
            Dim ftp As FTPClient = New FTPClient(Protocol,
                          Host, User, Pass, SSL, Passive)
            Dim i As Integer = ftp.Download(BaseFilePath
                             + filename, FileDestination)

            'Se ho un errore, cancello il file e inserisco un
                                           log di errore
            If (i <> 0) Then

                File.Delete(FileDestination)
                Dim sw As StreamWriter =
```

```
                                        File.AppendText(ErrorsFile)
                Try
                    sw.WriteLine(DateTime.Now & " " &
                                    FTPClient.GetErrorMessage(i))
                Finally
                    sw.Close()
                End Try
            End If

        ElseIf Protocol = "share" Then

            Dim f As FileInfo = New FileInfo(BaseFilePath
                                                & filename)
            If (f.Exists) Then
                f.CopyTo(FileDestination,True)
            Else
                File.Delete(FileDestination)
            End If

        End If

    End Sub
```

In sequenza:

- vengono lette le variabili dall'app.config
- viene generato il nome del file da scaricare in formato standard
- se il metodo è ftp, viene istanziata la classe FTPClient e richiamato il metodo Download
- se il metodo è share, il file è prelevato dal percorso di rete

Eseguendo il programma, il file dei log del giorno precedente alla data di esecuzione verrà scaricato nella cartella di destinazione.

IMPORTIAMO I DATI!

Occupiamoci adesso di importare i dati nelle tabelle di SQL Server utilizzando un pacchetto SSIS (SQL Server Integration Services).
Apriamo Visual Studio e creiamo un nuovo progetto di tipo "Business Intelligence" chiamato IISLogImport.

Figura 5: Creiamo il package SSIS

Come prima cosa, inseriamo un "Execute Process Task", chiamiamolo "FTP Download" ed associamo ad esso l'esecuzione del programma che effettua il download del file di log. Creiamo sul disco C una cartella chiamata C:\IISLogs e copiamo in essa l'eseguibile ottenuto dalla compilazione del precedente progetto. Configuriamo quindi il task: clickiamo sull'icona con il tasto destro del mouse e quindi su "Edit". Nel tab "Process" scriviamo "C:\IISLogs\IISLogFTP Download.exe" in corrispondenza della voce "Executable". Aggiungiamo quindi un "Data Flow Task" che chiameremo "Import Task".
Facciamo doppio clic sull'icona del Data Flow Task. Così facendo passiamo nella parte di definizione del DataFlow. Introduciamo innanzitutto la sorgente dati, ovvero il file che abbiamo scaricato in "C:\IISLogs\ IISLog.txt".Inseriamo nel Data Flow una "Flat File Source", clicchiamo sopra l'icona con il tasto destro del mouse e selezioniamo "Edit". Nella sezione "Connection Manager" clicchiamo su "New…" per collegarci alla sorgente dati. Nella

Figura 6: La prima parte del package

maschera di creazione della sorgente dati, diamo un nome alla sorgente e selezioniamo il file IISLog.txt. Spostiamoci quindi nella sezione "Columns" ed inseriamo "{CR}{LF}" come delimitatore di riga ed

Figura 7: Le colonne del file sorgente

Figura 8: Deriviamo la colonna data-ora

uno spazio come delimitatore di colonna; questo perché nei file di log di IIS le colonne sono delimitate per l'appunto da un carattere blank. Nella sezione "Advanced" impostiamo i nomi delle colonne, i tipi di dato e le dimensioni.
Il file di log di IIS contiene due campi separati per la

Figura 9: Mappiamo i campi del file di testo con le colonne della tabella

data e per l'ora; nella nostra tabella di SQL Server abbiamo un solo campo "data-ora", aggiungiamo quindi un task che ci permetta di derivare la colonna data-ora a partire dalle due colonne separate.
Il task in questione è il "Derived Column". Inseriamolo nel Data Flow. Al solito, clicchiamo sopra l'icona con il tasto destro del mouse e quindi su "Edit" per configurarlo. Aggiungiamo una nuova colonna derivata: "log_datetime" e come espressione inseriamo il seguente codice:

```
log_date + " " + log_time
```

Figura 10: Gli oggetti del Data Flow

A questo punto non ci resta che configurare la destinazione dei dati. Inseriamo una "OLE DB Destination" e configuriamola.
Nella sezione "Connection Manager", selezioniamo il database e quindi la tabella di destinazione dei dati. Nella sezione "Mappings" configuriamo le corrispondenze tra le colonne del file

Figura 11: Scheduliamo l'esecuzione del package

sorgente e quelle della tabella di destinazione. A questo punto il package è pronto, non ci resta che importarlo in SQL Server. Innanzitutto. creiamo l'unità di Deploy. Clickiamo con il tasto destro del mouse sul nome della solution e selezioniamo "Properties". Impostiamo a True il valore del parametro "CreateDeploymentUtility" e compiliamo il progetto. Andiamo nella directory "Bin" in cui troveremo il file "IISLogImport.dtsx". Copiamo il file "C:\IISLogsDeploy", apriamo una commad prompt ed eseguiamo:

```
dtutil /FILE " C:\ IISLogsDeploy \ IISLogImport.dtsx"
    /DestServer Nome_Server /COPY SQL; IISLogImport
```

Non ci resta che schedulare l'esecuzione del package. Apriamo "SQL Server Management Studio", colleghiamoci al server ed espandiamo il menu "SQL Server Agent". Clicchiamo con il tasto destro del mouse su "Jobs" e clicchiamo su "New job…"
Diamo un nome al package. Nella sezione "Steps" aggiungiamo lo step che permette di eseguire il package. Come "Type" scegliamo "SQL Server Integration Services Package", scegliamo quindi il package che abbiamo appena importato in SQL

Table - dbo.log_IISLogs | CRI\SQL2005.I...QLQuery1.sql* | Summary

log_id	log_datetime	log_client_ip	log_method	log_uri	log_uri_query	log_sta_status	log_user
248	10/02/2007 16.43.24	127.0.0.1	GET	/iislogdemo/chisiamo.htm	-	401	Mozilla/4.
249	10/02/2007 16.43.24	127.0.0.1	GET	/iislogdemo/chisiamo.htm	-	304	Mozilla/4.
250	10/02/2007 16.43.24	127.0.0.1	GET	/iislogdemo/Immagine_02.png	-	304	Mozilla/4.
251	10/02/2007 16.43.26	127.0.0.1	GET	/iislogdemo/default.htm	-	304	Mozilla/4.
252	10/02/2007 16.43.26	127.0.0.1	GET	/iislogdemo/Immagine_01.png	-	304	Mozilla/4.
253	10/02/2007 16.43.26	127.0.0.1	GET	/iislogdemo/dove.htm	-	304	Mozilla/4.
254	10/02/2007 16.43.26	127.0.0.1	GET	/iislogdemo/Immagine_04.png	-	304	Mozilla/4.
255	10/02/2007 16.43.27	127.0.0.1	GET	/iislogdemo/contatti.htm	-	304	Mozilla/4.
256	10/02/2007 16.43.27	127.0.0.1	GET	/iislogdemo/Immagine_03.png	-	304	Mozilla/4.
257	10/02/2007 16.43.34	127.0.0.1	GET	/iislogdemo/contatti.htm	-	200	Mozilla/4.
258	10/02/2007 16.43.34	127.0.0.1	GET	/iislogdemo/Immagine_03.png	-	200	Mozilla/4.
259	10/02/2007 16.43.48	127.0.0.1	GET	/iislogdemo/default.htm	-	401	Mozilla/5.
260	10/02/2007 16.43.59	127.0.0.1	GET	/iislogdemo	-	401	Mozilla/4.
261	10/02/2007 16.43.59	127.0.0.1	GET	/iislogdemo/	-	302	Mozilla/4.
262	10/02/2007 16.43.59	127.0.0.1	GET	/iislogdemo/Default.htm	-	304	Mozilla/4.
263	10/02/2007 16.43.59	127.0.0.1	GET	/iislogdemo/Immagine_01.png	-	304	Mozilla/4.
264	10/02/2007 16.44.01	127.0.0.1	GET	/iislogdemo/default.htm	-	304	Mozilla/4.
265	10/02/2007 16.44.01	127.0.0.1	GET	/iislogdemo/chisiamo.htm	-	304	Mozilla/4.
266	10/02/2007 16.44.01	127.0.0.1	GET	/iislogdemo/Immagine_02.png	-	304	Mozilla/4.
267	10/02/2007 16.44.01	127.0.0.1	GET	/iislogdemo/dove.htm	-	304	Mozilla/4.

Figura 12: I dati su SQL Server!

Server.Passiamo quindi alla sezione "Schedules" per configurare l'esecuzione giornaliera del package. Inseriamo "Daily" come frequenza e l'una di notte come orario per l'esecuzione.

NOTA

CODICI DI STATO

Quando viene effettuata una richiesta ad un web server, questo, oltre a fornire la risposta,restituisce dei codici di stato che indicano che esito ha avuto la richiesta.

Sono raggruppati nel seguente modo:

1xx – Messaggi informativi
2xx - Esito positivo
3xx - Reindirizzamento
4xx - Errore del client
5xx - Errore del server

La lista completa con le descrizioni è presente nella tabella "sta_status_codes" del database contenuto nel CD allegato

Salviamo la schedulazione. Per provare l'importazione, clicchiamo con il tasto destro del mouse sul nome del job che abbiamo salvato e selezioniamo "Start Job". Se tutto è andato a buon fine, dovremmo trovare i dati nella nostra tabella.

VISUALIZZIAMO I DATI

A questo punto i dati sono su SQL Server, "non ci resta che" preparare un'applicazione web per visualizzarli… Creiamo un progetto web chiamato "IISLogViewerVB" ed aggiungiamo il web.config contente la stringa di connessione al database ed una "MasterPage" che farà da template per la nostra applicazione. Il codice della master page è presente nel CD allegato.
La pagina ha tre componenti principali:

- una TreeView per i link alle funzioni
- un ContentPlaceHolder per la lista dei dati
- un ContentPlaceHolder per i grafici

Una volta creata la MasterPage, occorre realizzare le pagine dei report. Prima di procedere alla realizzazione delle pagine, creiamo una classe "Utils.vb" che conterrà i metodi per popolare il GridView dei dati e generare il grafico relativo.
La classe contiene due funzioni Shared:

Figura 13: I dati su SQL Server!

- LoadGrid: per caricare la griglia
- LoadGraph: per generare il grafico

Analizziamole una alla volta.

```
Public Shared Function LoadGrid(ByVal SQL As String,
                    ByVal grd_dati As GridView) As DataSet

    Dim Result As DataSet = New DataSet()

    Dim conn As SqlConnection = New
  SqlConnection(ConfigurationManager.ConnectionStrin
            gs("ConnectionString").ConnectionString)
    Try

        conn.Open()
        Dim Command As SqlCommand = New
                            SqlCommand(SQL, conn)
```

```
        Dim da As SqlDataAdapter = New
                         SqlDataAdapter(Command)

        da.Fill(Result)
        grd_dati.DataSource =
                       Result.Tables(0).DefaultView
        grd_dati.DataBind()

    Finally

        conn.Close()
    End Try

    Return Result

End Function
```

La funzione popola un GridView (passato come parametro) a partire da una query e restituisce un DataSet.
Vediamo adesso la procedura per caricare il grafico:

```
Public Shared Sub LoadGraph(ByVal Result As
     DataSet, ByVal ImagePath As String, ByVal FileName
           As String, ByVal SessionID As String, ByVal
   img_graph As System.Web.UI.WebControls.Image)

    Dim max As Double = 0
    Dim Values(Result.Tables(0).Rows.Count) As
                                          Integer
    For i As Integer = 0 To
                    Result.Tables(0).Rows.Count - 1
        Values(i) =
    Convert.ToInt32(Result.Tables(0).Rows(i).ItemArray
                                             (3))
        If Values(i) > max Then
            max = Values(i)
        End If
    Next

    Dim spaces As Integer = 35
    Dim scale As Double = (100 / max)

    Dim intMax As Integer = 100
    Dim intScale As Integer =
                  Convert.ToInt32(Math.Floor(scale))

    Dim bm As Bitmap = New Bitmap(400, intMax)
    Dim g As Graphics
    g = Graphics.FromImage(bm)
    g.Clear(Color.White)

    For i As Integer = 0 To Values.Length - 1
        g.FillRectangle(New SolidBrush( _
            Color.FromArgb(113, 248, 110)), _
            (i * spaces) + 15, _
            intMax - Convert.ToInt32((Values(i) *
                                       scale)), _
```

L'AUTORE

È un ingegnere informatico. Si occupa di sviluppo software in ambiente .Net per una società di informatica di Milano. Gestisce un sito ricco di script e manuali per chi si affaccia al mondo della programmazione web (www.morpheusweb.it).

```
            20, _
            Convert.ToInt32((Values(i) * scale)) + 5)
        g.DrawRectangle( _
            Pens.Black, _
            (i * spaces) + 15, _
            Convert.ToInt32(intMax - (Values(i) *
                                        scale)), _
            20, ( _
            Convert.ToInt32(Values(i) * scale)) + 5)
    Next
    bm.Save(ImagePath + "\\" + FileName +
                SessionID + ".jpg", ImageFormat.Jpeg)
    img_graph.ImageUrl = "~\\graph\\" + FileName
                                + SessionID + ".jpg"
End Sub
```

Utilizziamo il DataSet restituito dalla funzione LoadGrid per leggere i valori trovati con la query ed inserirli in un'array. Con:

```
Dim bm As Bitmap = New Bitmap(400, intMax)
```

creiamo una bitmap di dimensioni 400x100 che fungerà da "placeholder" per il nostro grafico. Cicliamo quindi l'array dei valori e, per ciascun elemento, disegniamo un rettangolo le cui dimensioni sono proporzionate rispetto al massimo valore trovato. Salviamo infine l'immagine e la associamola all'oggetto "img_graph" che sarà presente nella pagina aspx. Una volta create le funzioni base, passiamo alla realizzazione dei report. Prepariamo la WebForm delle pagine viste. Aggiungiamo una pagina che eredita dalla MasterPage e chiamiamola "PagineViste.aspx". Nei ContentPlaceHolder aggiungiamo un GridView e, in basso, un componente Image per il grafico. Analizziamo quindi la query per recuperare i dati. Prima, però, occorre ricordare che IIS logga l'accesso a qualsiasi risorsa; quindi, nei log troveremo, oltre alle righe relative alle pagine, anche le richieste relative ad immagini, icone ed altri tipi di file. In un report delle pagine viste siamo interessati ad un determinato tipo di risorsa (html, asp, aspx ect…). Per poter filtrare le tipologie di documenti che ci interessano, abbiamo in precedenza creato la tabella ext_extensions, che contiene l'estensione del file e il tipo di documento relativo su cui poter fare dei filtri.

```
SELECT
  YEAR(log_datetime) AS Anno, MONTH(log_datetime)
                AS Mese, DAY(log_datetime) AS Giorno,
  COUNT(*)
 FROM
 (
  SELECT *,  SUBSTRING(
       log_uri, CHARINDEX('.',log_uri, LEN(log_uri)- 5)+1,
       LEN(log_uri)-CHARINDEX('.',log_uri, LEN(log_uri)-5)
   ) AS Extension
```

```
  FROM log_IISLogs (NOLOCK)
 ) AS IISLogs
 INNER JOIN ext_extensions
  ON ext_extension = IISLogs.Extension
 WHERE ext_type = 'PAG'  GROUP BY
  YEAR(log_datetime), MONTH(log_datetime),
                                DAY(log_datetime)
 ORDER BY
  YEAR(log_datetime), MONTH(log_datetime),
                                DAY(log_datetime)
```

Abbiamo una sub-query che estrae l'estensione della risorsa che è stata richiesta. Mettiamo il risultato della query in join con la tabella delle estensioni e recuperiamo esclusivamente le righe il cui tipo è stato codificato come "PAG", ovvero pagina.

```
Protected Sub Page_Load(ByVal sender As Object,
        ByVal e As System.EventArgs) Handles Me.Load
    Try
        Dim SQL As String =  " SELECT" & _
            "  YEAR(log_datetime) AS Anno,
    MONTH(log_datetime) AS Mese, DAY(log_datetime)
                                  AS Giorno," & _
            "  COUNT(*)" & " FROM" & _
            " (" & _"  SELECT" & _
            "    *," & _"    SUBSTRING(" & _
            "      log_uri, CHARINDEX('.',log_uri,
                              LEN(log_uri)-5)+1, " & _
            "      LEN(log_uri)-CHARINDEX('.',log_uri,
                                   LEN(log_uri)-5)" & _
            "   ) AS Extension " & _
            "   FROM log_IISLogs (NOLOCK)" & _
            " ) AS IISLogs" & _
            " INNER JOIN ext_extensions" & _
        "  ON ext_extension = IISLogs.Extension " & _
            " WHERE ext_type = 'PAG'" & _
            " GROUP BY " & _
            "   YEAR(log_datetime),
      MONTH(log_datetime), DAY(log_datetime)" & _
            " ORDER BY" & "   YEAR(log_datetime),
          MONTH(log_datetime), DAY(log_datetime)"
        Dim Result As DataSet = Utils.LoadGrid(SQL,
                                          grd_dati)
        Utils.LoadGraph(Result,
   Server.MapPath("graph"), "visite", Session.SessionID,
                                         img_graph)
    Catch
        'gestione errori
    End Try
End Sub
```

A questo punto il gioco è fatto, non ci resta che inserire la query nella pagina e richiamare i due metodi per riempire la griglia e disegnare il grafico, eseguire il progetto e goderci le nostre statistiche in tutto il loro splendore!

Carmelo Scuderi

UNA SOLUZIONE AJAX PER IL DRAG & DROP

IMPAREREMO COME CREARE UN CARRELLO DELLA SPESA CHE ACCETTA I PRODOTTI SEMPLICEMENTE PER TRASCINAMENTO. PER IMPLEMENTARE IL TUTTO SARÀ SUFFICIENTE QUALCHE RIGA DI JAVASCRIPT E UN PO' DI HTML AVANZATO

In questo articolo creeremo una web application dotata di un carrello della spesa dove potremo inserire i prodotti semplicemente trascinandoli dalla loro posizione iniziale.
Al termine dell'operazione di trascinamento, verrà effettuata una chiamata asincrona al server, comunicandogli l'avvenuto acquisto del prodotto, con conseguente aggiornamento del database.
Faremo tutto questo utilizzando Ajax e Javascript. L'accoppiata ci consentirà di non effettuare mai alcun reload della pagina, ed utilizzare l'applicazione web come faremmo normalmente con un'applicazione dekstop.
L'utilizzo di Ajax, per coloro che non conoscessero ancora questa tecnica di progettazione *web-based*, ci permette di interagire con il server in qualsiasi momento senza dover per forza "ricaricare" la pagina corrente: il server, non appena ricevuta la richiesta via POST o GET, restituirà al client il risultato di un'elaborazione (nel nostro caso, l'aggiornamento del DB). Questo risultato verrà processato da una funzione Javascript chiamata in modo asincrono, che aggiornerà il contenuto del DIV "carrello" utilizzando le funzionalità del DOM (Document Object Model).
È evidente che l'utilizzo di Ajax ci agevola moltissimo, visto che possiamo lavorare direttamente nella nostra pagina web "vecchia", che viene di volta in volta aggiornata dalle transazioni che effettuiamo con il server in modo asincrono e trasparente.
A questo punto occorre fare, tuttavia, una precisazione: le funzionalità di drag & drop, anche abbinate ad Ajax, fanno già parte, spesso, di molti framework, i quali consentono, peraltro, di integrare le stesse con effetti speciali, animazioni, e così via, in modo estremamente semplice ed intuitivo. Va rimarcato, peraltro, che l'utilizzo diretto di Ajax e Javascript senza l'utilizzo di framework specializzati è una scelta da valutarsi attentamente, a seconda della grandezza e complessità dell'applicazione da realizzare.

COME INIZIARE

Sul CD allegato alla rivista è disponibile il codice sorgente del presente articolo, diviso in due cartelle. Per eseguire il carrello occorre disporre di un'installazione PHP – MySql, e di un web-server. Andrà quindi creato un DB ed eseguito il file SQL allegato, quindi bisognerà copiare la cartella all'interno del web-server. Per eseguire il gioco, allo stesso modo, occorre copiare la cartella relativa all'interno del web-server.

Detto questo, bisogna comunque considerare che non sempre le nostre esigenze si adattano perfettamente alle funzionalità offerte da un framework, mentre spesso conviene molto di più scrivere noi direttamente il codice che ci interessa, per risparmiare una grande quantità di spazio e raggiungere effettivamente l'obiettivo desiderato.
Nel nostro esempio, infatti, vogliamo sì dotare di funzionalità drag & drop la nostra applicazione, ma vogliamo nello stesso tempo ottenere che il trascinamento non coinvolga direttamente il prodotto su cui abbiamo cliccato, ma una sua copia generata dinamicamente sul momento.
Ecco, quindi, che nel caso in questione, evidentemente, ci troviamo di fronte ad una richiesta molto particolare, che mal si adatta ad essere risolta con un framework, per versatile e completo che sia.
In questi casi, peraltro, ci viene incontro il già nominato DOM, che rappresenta un modello di definizione del codice XHTML basato su una struttura ad albero, in cui ogni nodo corrisponde ad un elemento della pagina web. La struttura a nodi permette, quindi, di inserire i singoli elementi all'interno di una vera e propria gerarchia basata sull'oggetto Document, ovvero il "contenitore" principale della pagina web, identificato dal tag HTML.
Tramite il DOM, in particolare, possiamo referenziare elementi esistenti o generarne di nuovi,

semplicemente utilizzando metodi standard, validi per qualsiasi browser.
La seguente porzione di codice crea degli elementi XHTML e li "appende" ad un DIV:

```
var trascina=document.getElementById("trascina");
immagine = document.createElement("img");
immagine.src="immagine.png";
immagine.style.cursor="move";

br=document.createElement("br");

testo=document.createTextNode("testo");

trascina.innerHTML=""; //svuoto il DIV
        preventivamente, usando il metodo innerHTML
trascina.appendChild(immagine); //comincio ad
        appendere i singoli elementi al DIV
...
```

Come si può vedere, il DOM consente un approccio al documento Web decisamente modulare, risparmiando allo sviluppatore parecchi grattacapi; per contro, la sintassi dei vari comandi è fatta in modo tale da generare righe di codice molto lunghe, che necessitano di particolare attenzione per essere interpretate correttamente.
Il funzionamento dell'applicazione può essere suddiviso nelle seguenti fasi:

1 innanzitutto, il server tramite PHP genera il contenuto della tabella contenente i prodotti da acquistare. Ho usato una tabella per impaginare con semplicità i prodotti, ma potevo usare qualsiasi altro contenitore (un DIV, un fieldset, o meglio ancora una lista (UL)).
Nell'esempio, tuttavia, mi sono limitato a creare staticamente il codice HTML della tabella, anche se è evidente che in un caso reale ciò dovrebbe essere effettuato proprio dal server (d'altra parte, la pagina ha già estensione PHP, quindi il lettore che desidera aggiungere codice lato server alla pagina può farlo agevolmente !).

2 le immagini dei prodotti hanno poi, sull'evento onmousedown, agganciata una funzione che riempie il DIV "trascina" con il contenuto del prodotto e lo rende visibile (il valore della proprietà Display, infatti, inizialmente è settato a "none"). Quest'ultimo DIV, peraltro, segue il mouse in ogni suo movimento, almeno finché non viene rilasciato il pulsante per terminare il trascinamento (quando, cioè, su onmouseup, l'oggetto globale che rappresenta il DIV viene deallocato e il DIV viene reso nuovamente invisibile).

3 Terminato il trascinamento, su rilascio del pulsante del mouse, viene eseguito un controllo per verificare se l'oggetto da noi trascinato si trova ora localizzato all'interno del DIV "carrello". In tal caso, viene contattato tramite Ajax il server, che aggiorna il DB, e restituisce il totale dei nostri acquisti. Se anche questa funzione va correttamente in porto, allora viene creata una nuova lista UL riferita al prodotto in questione (sempreché questa già non esista) e viene aggiunto un numero di elementi LI ("list-item") pari al contenuto numerico della casella di testo del DIV "trascina". Se la lista già esiste, invece, allora la funzione semplicemente "appende" i prodotti ad essa.

IL WEB E LA TRASPARENZA

Il Drag & Drop si presta moltissimo ad essere arricchito con effetti grafici particolarmente accattivanti: uno di questi è sicuramente la trasparenza. Il formato delle immagini che più si presta a questo tipo di utilizzo è sicuramente il PNG (Portable Network Graphics), nato nel 1995 come alternativa al GIF, che all'epoca era coperto da brevetto. Il PNG, in particolare, ha caratteristiche molto migliori del GIF, che vanno da un miglior rapporto di compressione, al supporto della trasparenza variabile e di una gamma di colori molto più ricca. Di contro, Internet Explorer fino alla versione 6 non supporta correttamente la trasparenza con il PNG, aggiungendo alle immagini uno sfondo grigio. Per ovviare a questi inconvenienti Microsoft ha messo a disposizione diversi strumenti, tra cui filtri proprietari, *behaviours* (comportamenti particolari associati a determinate regole CSS, che nel caso in questione si basano su un file .htc e una gif trasparente che devono essere inseriti nella cartella dell'applicazione), ecc...

GESTIAMO IL TAG DIV

La posizione del DIV "trascina", come già anticipato più sopra, è vincolata al puntatore del mouse per tutta la durata del trascinamento. In particolare, non appena viene premuto il pulsante del mouse sulla figura, una funzione calcola la posizione corrente del puntatore. Tale posizione, naturalmente, deve rimanere costante, rispetto a quella del DIV, per tutta l'operazione di trascinamento; altrimenti, appena si comincia a muovere il puntatore, questo andrebbe immancabilmente a posizionarsi sull'angolo superiore sinistro del DIV. È necessario quindi all'inizio, calcolare anche la posizione precisa del DIV, e quindi effettuare una correzione continua rispetto alla posizione del puntatore del mouse, per tutta la durata del trascinamento.
La posizione precisa del DIV, tuttavia, non coincide necessariamente con le proprietà offsetLeft e offsetTop, visto che queste ultime rappresentano le distanze rispettivamente da

sinistra e dall'alto ma **solo rispetto all'elemento contenitore**. In definitiva, per ottenere la posizione precisa di un elemento all'interno della pagina, occorre sommare tutti gli offsetLeft (e ovviamente gli offsetTop) dei vari elementi, rispetto ai relativi contenitori. Tale operazione è svolta dalla seguente funzione:

```
function trova_posizione(elemento){
    var distanza_orizzontale = 0;
    var distanza_verticale  = 0;

    while (elemento.offsetParent){
        distanza_orizzontale += elemento.offsetLeft;
        distanza_verticale  += elemento.offsetTop;
        elemento      = elemento.offsetParent; //la
 variabile elemento, ora, passa a rappresentare il
//contenitore dell'elemento finora preso in
considerazione
    }

    return {x:distanza_orizzontale,
     y:distanza_verticale}; //ritorno le proprietà x e y,
}
```

Questa accortezza non è necessaria, evidentemente, quando l'elemento di cui si vuole calcolare la posizione precisa non si trova all'interno di altri elementi che fungono da contenitori: ciò avviene, in particolare, nella seconda applicazione di esempio, rappresentata da una semplice (e solo abbozzata) riproduzione del gioco della dama, dove esiste, tuttavia, una singola pedina, che può essere trascinata per tutta la superficie della pagina. La posizione precisa della pedina viene, poi, ricalcolata di volta in volta dal server al termine del suo trascinamento, sempre a mezzo di chiamate asincrone Ajax.
A titolo di esempio, ecco qui di seguito la funzione che corregge la posizione della pedina per farla rientrare perfettamente nelle caselle bianche:

```
function gestisci_risposta() {


    if(http.readyState == 4 || http.readyState==0)
    {

        try
        {
            var risposta =
                http.responseXML.documentElement;

            var val_nuovo_x, val_nuovo_y, stringa;
                val_nuovo_x=
 risposta.getElementsByTagName("x").item(0).firstChi
                                  ld.data.toString();
```

QUALI LIBRERIE MI POSSONO AIUTARE PER CREARE EFFETTI GRAFICI AVANZATI?

Come già citato nel testo dell'articolo, esistono parecchi framework che forniscono funzionalità molto avanzate, anche basate su Ajax, con poco sforzo. A titolo di esempio posso citare *script.aculo.us*, usato spesso con il framework *prototype*, ma ne esistono anche molti altri. Un framework decisamente complesso e ricco di funzionalità molto interessanti è poi *Dojo*, che fornisce controlli Ajax decisamente avanzati, strumenti per generare "al volo" grafici, menu, ecc...

```
                val_nuovo_y =
 risposta.getElementsByTagName("y").item(0).firstChi
                                  ld.data.toString();
                stringa =
 risposta.getElementsByTagName("ancorato").item(0).
                         firstChild.data.toString();

                trascina.style.left=val_nuovo_x;
                trascina.style.top=val_nuovo_y;
                trascina.setAttribute('ancorato', stringa);

                trascina = null;
        }
        catch(e)
        {
            alert("Attendere qualche secondo per
              effettuare la richiesta ..."  + e.toString());
        }

    }

}
```

CONCLUSIONI

L'avvento di Ajax sta letteralmente cambiando il modo di pensare al Web, tanto che alcune applicazioni come quelle che abbiamo appena che rappresentano la normalità in ambiente desktop appaiono come una rottura della consuetudine in ambiente web. Questo lascia intendere quanto la rivoluzione sia soprattutto concettuale

Enrico Viale

L'AUTORE

Enrico Viale è specializzato nella realizzazione di applicazioni Web o Desktop per privati o aziende. Per contattarlo, è possibile utilizzare il seguente indirizzo email: enrico.viale@gmail.com.

E' POSSIBILE FARE ANIMAZIONI SUL WEB SENZA USARE FLASH, JAVA, ECC... ?

Ovviamente la risposta è affermativa. È possibile ricorrere all'uso dei timer, attraverso ad esempio le funzioni *setInterval* (per temporizzare l'esecuzione di una determinata funzione) e *setTimeout* (per eseguire uno script dopo che è passato un certo intervallo di tempo): l'animazione riguarda evidentemente la posizione di un dato elemento che viene aggiornata dinamicamente dal timer stesso. Per rimuovere i timer, determinandone quindi l'arresto, viene invece impiegata la funzione *clearInterval*.

JAVASCRIPT CROSS-DOMAIN

IN QUESTO ARTICOLO IMPAREREMO AD UTILIZZARE JAVASCRIPT PER EFFETTUARE CHIAMATE ASINCRONE AD UN DOMINIO DIVERSO DA QUELLO DI APPARTENENZA. COME VEDREMO, QUESTA TECNICA CI PERMETTERÀ DI SPOSTARE IL CARICO DI LAVORO DAL SERVER AL CLIENT

Negli ultimi tempi si parla molto di Ajax. Per chi non lo sapesse Ajax sta per Asynchronous JavaScript and XML. Sostanzialmente è una tecnologia utilizzata per effettuare chiamate asincrone al server per poi aggiornare dinamicamente gli oggetti della pagina Web. Come tutte le tecnologie, però, anche Ajax ha i suoi vantaggi e svantaggi. Tra gli aspetti negativi di maggior rilievo vi è l'impossibilità di comunicare con un dominio diverso da quello di appartenenza utilizzando l'elemento principe della tecnologia Ajax, ossia l'oggetto XMLHttpRequest. In quest'articolo vedremo come sia possibile ovviare a tale problema. A riprova di ciò, svilupperemo una semplice applicazione che interroga uno dei Web service esposti da Yahoo! per recuperare le ultime news su un qualsiasi argomento scelto dall'utente. La **figura 1** mostra uno screenshot dell'applicazione di esempio.

Fig. 1: Screenshot relativo alla nostra applicazione.

IL PROBLEMA

Il problema di cui stiamo parlando viene identificato in inglese con i termini *cross-domain restriction*, ossia le impostazioni di sicurezza, insite nei moderni Web browser, sono tali da consentire chiamate al solo dominio da cui la pagina Web è stata generata. Ciò significa che utilizzando un linguaggio che gira lato client, come JavaScript, non potremmo effettuare chiamate a domini differenti da quello di appartenenza dello script. Ho detto "non potremmo" proprio perché, come vedremo, esiste un modo per aggirare questa fastidiosa restrizione. Essa, infatti, non ci permette (usando JavaScript) di utilizzare i vari servizi Web data la loro natura di trovarsi in domini completamente diversi. La **figura 2** dovrebbe chiarire la problematica oggetto della discussione.

Fig. 2: Problema della restrizione cross-domain.

Al problema illustrato, in realtà, esistono diverse soluzioni, tra le quali:

- Utilizzare mod_rewrite o mod_proxy di Apache per passare la richiesta dal nostro server a qualsiasi altro. Così facendo il client saprebbe di avere a che fare con lo stesso dominio da cui è arrivata l'applicazione JavaScript e quindi non ci sarebbero problemi. Apache si occuperebbe dello "sporco lavoro" di fare il redirect verso il server desiderato.
- Applicare una firma digitale agli script. Questa soluzione, però, non è molto praticabile se non altro perché, al momento in cui sto scrivendo, è supportata soltanto da Firefox.
- La soluzione che viene più spesso utilizzata è quella di usare un proxy. Il problema con questo tipo di approccio risiede nel fatto che bisognerebbe configurare il proxy in modo tale che, per

ragioni di sicurezza, non passi le richieste verso qualsiasi dominio. Inoltre, utilizzare un proxy, o il mod_rewrite di Apache non fa altro che aggiungere un anello alla catena che porta la nostra applicazione client a comunicare col server desiderato. Ciò comporta un abbassamento delle performance.

- L'approccio ideale sarebbe quello di trovare una soluzione che ci permetta di mettere in comunicazione diretta il client e il server senza intermediari. Guarda caso tale soluzione è oggetto del presente articolo. Le tecnologie sfruttate per raggiungere questo scopo sono JSON e la creazione dinamica del tag <script>. Quest'ultima va a sostituire l'oggetto XMLHttpRequest per effettuare le chiamate asincrone al server.

CREAZIONE DINAMICA DEL TAG <SCRIPT>

L'uso più comune che si fa del tag <script>, è di inserire pezzi di codice JavaScript all'interno di una pagina HTML o di includere un file JavaScript esterno. Questo tipo di utilizzo è statico. In altre parole, all'atto della scrittura della pagina HTML, inseriamo i tag <script> necessari a raggiungere i nostri scopi. Non tutti sanno, tuttavia, che è possibile utilizzare il DOM (Document Object Model) per includere il suddetto tag dinamicamente. Vediamo come ciò sia possibile. Supponiamo di avere una pagina HTML e di voler includere, in modo dinamico, uno script nella sezione <head> della stessa. Per effettuare tale operazione è sufficiente utilizzare il seguente codice JavaScript:

```
// punta alla sezione head
var headLocation = document.getElementsBy
                    TagName("head").item(0);
// costruisce lo script da inserire
var script = document.createElement("script");
script.type = "text/javascript";
script.src = "test.js";
// include lo script dinamicamente
headLocation.appendChild(script);
```

Come potete vedere non è niente di complicato. La prima riga di codice ottiene un riferimento al tag <head> del documento. In seguito viene costruito lo script da inserire di modo che abbia il seguente formato:

```
<script type="text/javascript" src="./test.js">
                                        </script>
```

Il file *test.js* contiene la singola istruzione:

```
alert("Ciao Mondo!");
```

L'ultima riga non fa altro che aggiungere lo script in oggetto come nodo figlio di <head>. Il risultato di tutto ciò è la visualizzazione di un alert contenente la stringa "Ciao Mondo!". Chiaramente avremmo potuto ottenere lo stesso risultato in modo più semplice. La potenza di ciò che abbiamo fatto sta nell'aver aggiunto lo script in modo dinamico e non staticamente. Quello che avviene quando viene incluso un tag <script> dinamicamente è che il suo codice (l'URL dello script) viene eseguito al volo. Per provarlo facciamo un altro esempio utilizzando JSON. Non entrerò nel dettaglio di JSON dato che ne abbiamo già parlato in diversi articoli per ioProgrammo. Potete, tuttavia, avere un assaggio su JSON leggendo il box laterale. Tornando al nostro esempio, supponiamo di avere una funzione che si aspetta in ingresso un oggetto in formato JSON e ne visualizzi la proprietà cognome. Tale oggetto avrà la seguente struttura:

```
{
"cognome" : "Lacava",
"nome" : "Alessandro"
}
```

Per chiarire un po' le cose avremo una pagina HTML così composta:

```
<html>
<head>
</head>
<body>
<script>
```

JSON IN BREVE

JSON sta per JavaScript Object Notation. Per chi avesse dei dubbi al riguardo, JSON non è né un linguaggio né una tecnica di programmazione. Esso è semplicemente un formato leggero di interscambio dati che utilizza la notazione letterale di JavaScript per rappresentare oggetti ed array. Sostanzialmente, quindi, è un sottoinsieme di JavaScript. Non entrerò nel dettaglio di JSON dato che gli è stato dedicato un intero articolo su ioProgrammo N. 108 – Novembre 2006. In breve, un oggetto, in JSON, è rappresentato con la seguente sintassi:

```
{
"cognome" : "Lacava",
"nome" : "Alessandro",
"anni" : 30
}
```

Il precedente esempio rappresenta, in modo letterale, l'oggetto Persona. Per rappresentare un array, invece, si usa la seguente sintassi:

```
{
  "linguaggi" : [
  "Java",
  "C#",
  "JavaScript"
 ]
}
```

Abbiamo così creato l'array linguaggi. Per interpretare un oggetto/array JSON in JavaScript si può utilizzare la funzione eval. Esistono anche diverse librerie lato server per mappare da linguaggi come PHP, Java e C# a JSON. Maggiori info su JSON li potete trovare sul sito ufficiale, all'indirizzo: http://www.json.org

Utilizzare un proxy per effettuare chiamate cross-domain con XMLHttpRequest: http://developer.yahoo.com/javascript/howto-proxy.html

Qui potete trovare un articolo interessante di Jason Levitt: http://www.xml.com/pub/a/2005/12/21/json-dynamic-script-tag.html

Documentazione sul servizio di ricerca news esposto da Yahoo!: http://developer.yahoo.com/search/news/V1/newsSearch.html

Qui, invece, trovate tutte le risorse necessarie che Yahoo! dedica ai programmatori JavaScript/Ajax: http://developer.yahoo.com/javascript/

```
// punta alla sezione head
var headLocation = document.getElementsByTag
                              Name("head").item(0);
// costruisce lo script da inserire
var script = document.createElement("script");
script.type = "text/javascript";
script.src = "./test.js";
// inserisce lo script dinamicamente
headLocation.appendChild(script);

// visualizza la proprietà cognome
function displayLastName(person)
{
        alert(person.cognome);
}
</script>
</body>
</html>
```

Inoltre, questa volta, il file test.js deve contenere il seguente codice:

```
displayLastName({"cognome" : "Lacava", "nome" :
                                  "Alessandro"});
```

Lanciando la pagina HTML notiamo che viene visualizzata la proprietà cognome, come ci aspettavamo. Questo conferma quanto detto in precedenza, in altre parole il codice dello script dinamicamente aggiunto viene eseguito "on the fly". Tale risultato è importantissimo poiché possiamo far puntare l'URL, inserito nell'attributo src di <script>, a qualsiasi dominio siamo interessati.Rimane un piccolo problema facilmente risolvibile. L'URL puntato deve restituire il risultato in formato JSON come argomento di una funzione di callback. Tale funzione la specifichiamo per mezzo di un parametro della query string. Fortunatamente Yahoo! fornisce questo tipo di servizio per alcuni dei suoi Web service. La parte restante dell'articolo si occuperà proprio dello sviluppo di un'applicazione JavaScript che invochi il Web service di Yahoo! relativo alla ricerca delle news dato un argomento. Per "complicare" (si fa per dire) un po' di più le cose l'applicazione includerà una combo box che permetterà di scegliere la lingua in cui cercare le news.

L'APPLICAZIONE DI ESEMPIO

La **figura 1** mostra uno screenshot dell'applicazione di esempio. Quest'applicazione è composta da:

- Un file HTML.
- Un foglio di stile rappresentato da un file CSS.
- Tre file JavaScript.

I primi due rappresentano la GUI mentre i file JavaScript hanno la responsabilità di recuperare le news e costruire il codice HTML da inviare alla GUI.
Vediamo tutto il giro dell'applicazione partendo dal singolo file HTML. Quello che segue è il codice della sezione <head> del file in questione:

```
<link rel="stylesheet" href="./style.css" />
<script src="./scripts/WebServiceRequester.js" type=
                              "text/javascript"></script>
<script src="./scripts/QueryPerformer.js" type="text/
                                   javascript"></script>
<script src="./scripts/ResultProcessor.js" type="text/
                                   javascript"></script>

<script type="text/javascript">
        // esegue la query (la chiamata al WS)
        function performRequest()
        {
                // svuota il contenuto del box
                                        delle news
                document.getElementById
                        ("newsBody").innerHTML = "";
                // recupera i dati del form
                var query = document.search
                                Form.searchText.value;
                var language = document.search
                            Form.searchLanguage.value;
                // istanzia il QueryPerformer
        passando il nome della funzione di callback
                // da richiamare quando la
                                chiamata al W.S. ritorna
                var queryPerformer = new
                QueryPerformer("wsProcessResult");
                // chiama il Web service
                queryPerformer.execute(query,
                                        language);
```

UN CHIARIMENTO SU AJAX

Si è parlato talmente tanto di Ajax in questo periodo che si è venuta a creare una grande confusione su cosa sia e a cosa serve. Ajax sta per Asynchronous JavaScript + XML. Il termine è stato coniato da Jesse James Garrett, presidente di Adaptive Path, in un articolo che potete trovare al seguente link:
http://www.adaptivepath.com/publications/essays/archives/000385.php
In sostanza, Ajax è un insieme di tecnologie utilizzate congiuntamente per sviluppare applicazioni, cosiddette, Web 2.0. Di quest'insieme di tecnologie fanno parte:

1. **(X)HTML e CSS, utilizzati per lo strato di presentazione.**
2. **DOM, per interagire dinamicamente col documento.**
3. **XML, come formato di interscambio dati.**
4. **XMLHttpRequest, per il recupero asincrono dei dati.**
5. **JavaScript, come "collante" delle precedenti.**

C'è da dire, tuttavia, che non è obbligatorio utilizzare XML come formato di interscambio dati. In alcuni contesti è preferibile usare JSON o qualche altro formato customizzato. A tal proposito occorre precisare che JSON non è alternativo ad Ajax ma può essere utilizzato in congiunzione a quest'ultima tecnologia come sostituto di XML per rappresentare i dati.

```
        }
</script>
```

Come si può vedere, il precedente codice "linka" il foglio di stile. In seguito include i tre file JavaScript che si occupano del recupero delle news. Dopodiché viene definita la funzione performRequest che avvia la ricerca delle news. Come avrete intuito, questa funzione viene chiamata non appena viene cliccato il bottone "Avvia Ricerca". Per tale motivo è inserita come valore dell'attributo action del form:

```
<form id="searchForm" name="searchForm"
                action="javascript:performRequest();">
```

La funzione performRequest recupera i dati del form, ovvero testo da cercare e lingua in cui si desiderano i risultati. In seguito istanzia un oggetto della classe QueryPerformer che utilizza per eseguire la query relativa alla ricerca. Per ragioni di spazio non sarà visto il foglio di stile che è abbastanza semplice. Lo potete, tuttavia, trovare nel codice allegato. Vediamo, invece, la parte più importante dell'applicazione, ovvero i file JavaScript. Quello che segue è il codice del costruttore di QueryPerformer:

```
function QueryPerformer(callback)
{
        this.searchUrl = "http://search.yahooapis.
            com/NewsSearchService/V1/newsSearch";
        this.appId = "YahooDemo";
        this.output = "json";
        // controlla che callback sia passata

        if(!callback || callback.length == 0)
        {
                throw new Error("La funzione di
                callback è un parametro obbligatorio!");
        }
        this.callback = callback;
}
```

Il parametro passato in ingresso rappresenta la funzione di callback da chiamare. In pratica è quella che riceverà il risultato della ricerca in formato JSON. Il costruttore, inoltre, definisce:

- **searchUrl:** l'URL che punta al Web service di Yahoo!
- **appId:** l'ID da utilizzare per la chiamata. Da notare che quello usato è un ID temporaneo. Conviene registrarsi sul sito di Yahoo! per ottenerne uno full.
- **output:** il formato in cui desideriamo i dati. Nel nostro caso JSON. Se tale parametro non è specificato, viene utilizzato come default il formato XML.

La classe QueryPerformer ha un solo metodo, execute, che viene invocato da performRequest:

```
QueryPerformer.prototype.execute = function(query,
                                        language)
{
    [...]
        // url completo
        var fullUrl = this.searchUrl + "?" +
                        "appid=" + this.appId +

                        "&query=" + query +
                        "&language=" + language +

                    "&output=" + this.output +
                        "&callback=" + this.callback;
        // effettua la chiamata al servizio
        var requester = new JSONscriptRequest
                                        (fullUrl);
        requester.buildScriptTag();
        requester.addScriptTag();
};
```

Questo metodo esegue alcuni semplici controlli che abbiamo omesso per praticità. In seguito costruisce l'URL completo che passa poi al costruttore della classe JSONscriptRequest. Tale classe è stata scritta da Jason Levitt. Non sarà analizzata per ragioni di spazio. Non è molto complessa e la potete in ogni caso trovare nel codice allegato. Vi basti sapere che tale classe è responsabile della costruzione dello script e la sua aggiunta dinamica alla sezione <head> del file HTML. La classe si trova nel file WebServiceRequester.js.
Rimane da esaminare la funzione di callback che viene invocata non appena la chiamata al Web service ritorna. Se ricordate, il nome di tale funzione è stato passato come parametro al costruttore di QueryPerformer. La funzione si aspetta un oggetto, in formato JSON, con la seguente struttura "imposta" dal servizio Web di Yahoo!:

```
{
        "ResultSet":
        {
                "totalResultsAvailable":"2824",
                "totalResultsReturned":10,
                "firstResultPosition":"1",
                "Result":
                [
                        {
                        "Title":"Exploit per
                    Vista? 50 mila dollari, grazie!",
                        "Summary":"Nel
                    mercato nero degli hackers...",
                "Url":"http:\/\/it.news.yahoo.
        com\/19122006\/255\/exploit-per-vista...",
                        "ClickUrl":"http:\/
            \/it.news.yahoo.com\/19122006\/255\
                            /exploit-per-vista...",
                "NewsSource":"Tom's Hardware
            Guide Italia via Yahoo! Italia Notizie",
```

```
                        "NewsSourceUrl":"http:\/\/it.
                                    news.yahoo.com\/",
                        "Language":"it",
                        "PublishDate":"1166508455",
                        "ModificationDate":"1166508552"
                        },
                        [...]
                ]
        }
}
```

Le proprietà di tale oggetto sono:

- **totalResultsAvailable:** indica il numero totale di risultati disponibili.
- **totalResultsReturned:** è il numero di risultati ritornati. Per default il servizio restituisce 10 risultati. È possibile modificare questo valore specificando il parametro results nella query string.
- **firstResultPosition:** indica la posizione del primo risultato. Anche questo parametro lo potete modificare agendo sui valori specificati nella query string. In un box laterale è indicato il link in cui potete trovare una descrizione completa dei parametri che è possibile utilizzare per personalizzare al massimo la query.
- **Result:** tale proprietà è un array dove ogni elemento rappresenta una news. Non entriamo nel dettaglio delle proprietà concernenti la singola news poiché autoesplicative.

La definizione della funzione di callback si trova nel file *ResultProcessor.js*. Eccone il codice:

```
function wsProcessResult(obj)
{
        var totalResultsReturned = obj.ResultSet.
                                        totalResultsReturned;
        if(totalResultsReturned == 0)
        {
                addNews("Nessuna news
                                        disponibile");
                return;
        }

        for(var i = 0; i < totalResultsReturned; i++)
        {
                // news corrente
                var news = obj.ResultSet.
                                        Result[i];
                // recupera i dati relativi alla news
                                        corrente
                var newsTitle = news.Title;
                var newsSummary = news.
                                        Summary;
                var newsUrl = news.ClickUrl;
                // costruisce l'HTML relativo alla
                                        news da visualizzare
                var newsToDisplay = "<a class=
        'title' target='_blank' href='" + newsUrl + "'>" +
                                        newsTitle + "</a>";
                newsToDisplay += "<br />";
                newsToDisplay += "<a class=
        'summary' target='_blank' href='" + newsUrl +
                                "'>" + newsSummary + "</a>";
                newsToDisplay += "<br />
                                        <br />";

                //aggiunge la news al blocco
                addNews(newsToDisplay);
        }
}
```

Questa funzione controlla se il numero di news ricevute è zero. In tal caso visualizza un messaggio adatto ed esce. La parte più interessante è, ovviamente, il loop. Esso cicla sull'array Result che, come abbiamo visto, contiene le news ricevute dal servizio. Per ogni news il loop recupera titolo, descrizione e url alla news completa. Con tali dati costruisce il codice HTML da inserire nel <div> (il cui id è newsBody) della pagina HTML. L'aggiunta vera e propria della news viene fatto dalla funzione addNews:

```
function addNews(newsToDisplay)
{
        document.getElementById("newsBody").
                                innerHTML += newsToDisplay;
}
```

Essa, semplicemente, appende l'HTML passato come parametro, ovvero la news, al corpo del div newsBody.

QUALI RESTRIZIONI?

Vi sono dei limiti, imposti da Yahoo!, per quanto riguarda il numero massimo di query che è possibile sottoporre al servizio. Per quanto riguarda il Web service delle news tale limite è fissato, al momento in cui scrivo, a 5000 query al giorno per ogni indirizzo IP. Per avere maggiori informazioni al riguardo vi consiglio di puntare il vostro browser al seguente URL: http://developer.yahoo.com/search/rate.html

CONCLUSIONI

Utilizzando gli script dinamici, è possibile sviluppare potenti applicazioni che recuperano dati da un Web service tramite l'uso del solo JavaScript. Il vantaggio principale di un approccio simile, risiede nel fatto che il nostro server non risente per niente del carico relativo alla comunicazione col servizio. Tutta la logica relativa al recupero, parsing e visualizzazione delle news, si trova lato client (JavaScript).

Daniele De Michelis

JYTHON: FONDE PYTHON CON JAVA

ECCO COME RACCOGLIERE IN UNICO LINGUAGGIO LA COMPLETEZZA DEL COMPILATORE DI SUN CON L'ELEGANZA E LA SEMPLICITÀ DI PYTHON. IN QUESTO NUMERO REALIZZEREMO ESEMPI COMPLESSI CON POCHISSIME RIGHE DI CODICE

Molte applicazioni cercano di coinvolgere l'utente usando una grafica accattivante, permettendo di ottenere una gran quantità di informazioni con pochi colpi di mouse. Anche il web, con AJAX, cerca di rendere la vita del navigatore più semplice e piacevole. In questo contesto si inserisce Jython, il fortunato incontro tra Python e Java: tra due colossi del genere non poteva che nascere uno strumento potente che sfrutta la flessibilità dell'uno e la ricchezza dell'altro. In questo articolo si daranno le basi per conoscere e usare Jython con molti esempi da sfruttare subito. L'articolo è diviso nelle due maggiori aree di utilizzo di Jython: le interfacce grafiche e i collegamenti al database. Come i libri di cucina, potete vedere i piatti da cucinare nelle foto e, di conseguenza, provare a realizzarli tutti in maniera sequenziale oppure passare a quello che vi sembra più appetitoso.

COME INIZIARE

Il primo passo è installare una versione del Java Development Kit. In allegato con al codice, trovate il file di installazione di Jython. È necessario eseguirlo come programma Java perché è un file .jar (su windows, c'è il riferimento usando il tasto destro). A questo punto gli esempi sulle GUI sono funzionanti. Per quelli relativi al db, è necessario installare MySQL o, in alternativa, PostgreSQL. Negli esempi, per praticità, si fa riferimento solo a MySQL; nel codice ci sono le istruzioni anche per PostgreSQL. Negli esempi sul db si fa riferimento al db chiamato IoP e a una tabella il cui script è presente in allegato (script.sql). Per eseguire gli esempi, serve andare nella directory nella quale è installato Jython (quella contente jython.bat), e copiare la directory IoP (che contiene gli esempi dell'articolo) e il file dbexts.ini (directory e file sono in allegato alla rivista). A questo punto, basta lanciare il comando jython IoP\banner.py e tutti gli altri esempi. Per visualizzare gli esempi proposti basta un semplice editor di testo o un tool di sviluppo. In questo caso, se si usa eclipse, c'è il plugin specifico per Jython che si trova al seguente indirizzo: http://www.redrobinsoftware.net/jydt/

Conoscenze richieste
Basi di programmazione Java, Python e di database

Software
JDK 1.4.1 o superiore, Jython, MySQL

Impegno

Tempo di realizzazione

INTERFACCE GRAFICHE UN SEMPLICE INIZIO

Per iniziare nel modo più semplice e intuitivo, si può partire con un po' di interfacce grafiche, *GUI, Graphical User Interfaces*, che mostrano immediatamente uno degli utilizzi che si può fare di Jython. I risultati dei nostri primi sforzi sono mostrati in **figura 1, 2** e **3**. Tanto per far capire l'estrema facilità del codice, si può già dire che le righe di codice del primo esempio sono 20 in tutto! E nonostante questo ci possono essere già dei primi spunti per riflettere sulle potenzialità di Jython.

```
# File: banner.py

from java import awt
import java # import delle classi java

class Banner(awt.Canvas):
    def paint(self, g):
        g.color = 204, 204, 204
        ...
        message = "Hello word!!!"
        ...
top_frame = awt.Frame("Jython Banner", size=
                                        (350, 150),
   windowClosing=lambda e:
                              java.lang.System.exit(0))
top_frame.add( Banner() )
top_frame.visible = 1
```

Come si può vedere dal codice, il commento della prima riga ricorda che il file è di tipo .py, cioè interpretabile da Python. Eppure, immediatamente sotto, l'istruzione di import è molto simile a quelle usate in Java. In poche parole, serve Jython che unisce la potenza di Java e la flessibilità di

Python. Un'altra considerazione che salta all'occhio è la definizione di class, che serve proprio a creare una classe, cioè la base della programmazione ad oggetti. L'esempio sfrutta Java anche per creare degli oggetti che appartengono al pacchetto grafico AWT, come Canvas. Il metodo paint può essere modificato a piacimento e con semplicità per cambiare i colori o il testo del messaggio. Le altre istruzioni servono per creare e rendere visibile la finestra creata. E fin qui si sono esplorati i confini del mondo Java. Python subentra per gestire facilmente la chiusura della finestra. Quindi, già questo primo esempio, semplice nella realizzazione e nel codice, lascia già intravedere le enormi possibilità che può offrire la fusione tra Java e Python.

Figura 1: ecco come appare la nostra prima applicazione

A questo punto si può passare all'esempio della **figura 2** che è la naturale evoluzione del precedente.

Figura 2: Aggiungere un bottone è un'operazione non complessa

```
# File: halloJythonGUI.py
...
# Funzione exit per chiudere la finestra
def exit(e):
    java.lang.System.exit(0)

# Definizione della finestra
top_frame = awt.Frame(title="Hallo",
                      background = awt.Color.yellow,
                      windowClosing=exit)

# Definizione del bottone e relativa azione
mybutton = awt.Button("OK",
                      actionPerformed=exit)

# Aggiunta del bottone
panel.add(mybutton)
...
```

La maggior parte delle istruzioni utilizzate in questo esempio, sono del tutto analoghe a quelle dell'esempio precedente. L'elemento aggiuntivo è il bottone, che viene inserito nella finestra. Il codice offre lo spunto per parlare una caratteristica usate da *Jython* per velocizzare e rendere più compatta la scrittura del codice. Nota con il nome di **Proprietà dei Bean**, consente di accedere alle variabili in modo diretto senza passare dai metodi get e set. Inoltre, si possono passare al costruttore i parametri come parole chiave. L'istruzione vista in precedenza, *awt.Frame(title="Hello", background = awt.Color.yellow, windowClosing=exit)* serve per creare un oggetto *Frame* e per impostare le sue proprietà. Quello che *Jython* fa in una riga di codice, Java lo farebbe con il codice seguente.

Figura 3: GUI con immagine

```
Frame myframe = new Frame("Hallo");
myframe.setBackground(Color.yellow);
myframe.addWindowListener(new
                SomeWindowListenerClass());
```

Per concludere questi primi esempi introduttivi, si può passare alla figura 3 che mostra una finestra contente un'immagine.

Il codice relativo offre un altro spunto.

```
# File immagine.py
...
if __name__ == '__main__':
    f = immagine("jython-new-small.gif")
```

Nel codice sono presenti nomi che iniziano e finiscono con "__". Ce ne possono essere di diverso tipo e sono variabili speciali. Nel nostro caso servono a dire che, se il chiamante è il *main* cioè si è lanciato a riga di comando proprio il file *immagine.py*, allora si eseguono le istruzioni successive, cioè l'inserimento dell'immagine. Per chi mastica *Java*, è l'equivalente di *public void main (String[] args)*.

DOVE TROVO MYSQL

Il popolare database può essere scaricato dal sito: http://dev.mysql.com/downloads/ Da questo indirizzo si può scegliere quale versione scaricare. Una è gratuita e non ha supporti tecnici, l'altra, a pagamento, è indicata per chi ha bisogno di aiuto. Tra gli strumenti utili per gestire il db, si segnalano i GUI Tools, programmi con interfacce semplici che consentono di creare e modificare il db.

GESTIONE DEGLI EVENTI

Dopo aver visto gli esempi introduttivi che spiegano le basi di Jython si può passare alla gestione degli eventi che, a dir la verità, risulta anch'essa semplice.
Le **figure 4** e **5** mostrano una finestra con un normale menù a tendina. Scegliendo una voce del menù, compare una scritta relativa all'operazione eseguita.

Figura 4: Menu di scelta

```
# File: menu.py
...
menus = [
   ('File', ['New', 'Open', 'Save', 'Saveas', 'Close']),
   ('Edit', ['Copy', 'Cut', 'Paste']),
]
...
class MenuTest(awt.Frame):
    def __init__(self):
        bar = awt.MenuBar()
        for menu, menuitems in menus:
            menu = awt.Menu(menu)
            for menuitem in menuitems:
                method = getattr(self, 'on%s' %
                                                  menuitem)

                item = awt.MenuItem
                (menuitem, actionPerformed=method)
                menu.add(item)
            bar.add(menu)
        self.menuBar = bar
        self.windowClosing = lambda e:
                                          System.exit(0)
        self.eventLabel = awt.Label("Event: ")
        ...
        self.add(self.eventLabel)

    def onNew(self, e):
        self.eventLabel.text = "Event: onNew"
```

In questa parte di codice c'è la dichiarazione della matrice che costituisce il menu e una scansione di tutti i suoi elementi per associare a ogni voce l'azione da eseguire. Vi è poi l'istruzione semplice e compatta *System.exit(0)* per utilizzare la chiusura della finestra con *Jython*. Infine c'è l'esempio di una funzione che viene chiamata quando si sceglie una voce dal menù.
In questo blocco di codice si può fare una considerazione per velocizzare la scrittura del codice, che è la missione di *Python*. Infatti, per gestire gli eventi chiamati dal menù, basta usare una sola istruzione: *actionPerformed=method*. Generalizzando, la **Gestione degli eventi** con *Jython* è resa estremamente semplice e compatta. Altri esempi che illustrano gli eventi sono i bottoni per chiudere le finestre.
Con questi file si ha un'idea di cosa può fare *Jython* e di come lo si può utilizzare. I file successivi sono una serie di esempi che danno altri strumenti utili per la gestione delle interfacce grafiche.

Figura 5: Azione del menu

ESEMPIO DI GUI: AGENDA

Molte applicazioni devono essere testate inserendo dei parametri in ingresso e verificando il comportamento dell'applicazione. Alcuni programmi servono per la gestione del personale. Prendendo spunto dall'esempio successivo, si può creare un'agenda o, in generale, un form nel quale inserire i dati. Anche in questo caso l'operazione non è particolarmente complessa.
Il risultato è mostrato in **figura 6** e di seguito ci sono le parti di codice più utili.

Figura 6: Ecco come apparirà la nostra agenda al termine del lavoro di programazione

```
# File: agenda.py
from pawt import GridBag
```

```
...
bag = GridBag(pane, fill="HORIZONTAL")
map(category.add, ["Family", "Friends",
                                          "Business"])
bag.add(labelFactory("Category: "))
bag.addRow(category, anchor="WEST",
                                          fill="NONE")
...
address = awt.TextField(35)
bag.add(labelFactory("Address: "))
bag.addRow(address)
```

Oltre alla definizione dell'array direttamente dove serve, è da notare una classe specifica di Jython per la gestione delle interfacce: GridBag. Questa classe si può utilizzare come elemento che gestisce le altri classi Java semplificandone l'utilizzo.

REALIZZIAMO UN ALBERO

Spesso capita di dover mostrare la mappa del proprio sito, oppure di dover illustrare le directory del file system. Il codice del prossimo esempio calza benissimo per questi scopi, come mostrato in **figura 7**.

Figura 7: Ecco la rappresentazione dell'albero

Di seguito vengono riportate le istruzioni più significative.

```
# File: albero.py
top_frame = swing.JFrame("Albero",
                    windowClosing=lambda e:
                                   sys.exit(0),
                    background=(180,180,200), )
data = tree.DefaultMutableTreeNode("Radice")
data.add(tree.DefaultMutableTreeNode("Foglia1"))
childNode = tree.DefaultMutableTreeNode("Nodo")
childNode.add(tree.DefaultMutableTreeNode("Fogli
                                          a2"))
data.add(childNode)
t = swing.JTree(data)
```

Ormai molti concetti sono noti: proprietà dei bean e gestione degli eventi nel costruttore. Gli elementi aggiuntivi sono specifici per la gestione degli alberi e sono la creazione della radice, dei nodi e delle foglie.

ESEMPIO DI GUI: DRAG & DROP

Una delle caratteristiche delle interfacce grafiche che serve a semplificare la vita dell'utente è senza dubbio il *Drag & Drop*. Con un semplice trascinamento del mouse, chiunque può spostare un elemento da una parte all'altra della finestra senza dover eseguire operazioni complicate.
Le **figure 8** e **9** mostrano la situazione prima e dopo lo spostamento.

Figura 8: Drag and drop prima

Figura 9: Drag and Drop dopo

```
# File: dragNDrop.py

from java.awt import dnd
...
class JythonDnD(awt.Frame,
   dnd.DragSourceListener,
                        dnd.DragGestureListener):
...
        self.draglist = awt.List()
        map(self.draglist.add, ["1","2","3","4","5"])

        self.droplist = droplist([])
        self.dropTarget = dnd.DropTarget(self,

 dnd.DnDConstants.ACTION_COPY_OR_MOVE,
                                   self.droplist)
...
    def dragGestureRecognized(self, e):
        item = self.draglist.getSelectedItem()
e.startDrag(self.dragSource.DefaultCopyDrop,
          datatransfer.StringSelection(item),self)
```

POSTGRESQL

In alternativa a MySQL si può utilizzare facilmente anche PostgreSQL, semplicemente cambiando i parametri di configurazione dell'url di connessione come riportato nel file dbexts.ini.
Il popolare database può essere scaricato dal sito:
http://www.postgresql.org/
A differenza di MySQL, è in linea tutta la documentazione scaricabile gratuitamente. Anche per questo db c'è la possibilità di usare delle comode interfacce grafiche.

Dal codice si può osservare l'utilizzo della classe *Java dnd* che semplifica tutto il processo grazie ai suoi metodi e alle costanti. E poi il metodo che si occupa di spostare gli elementi.
Anche questa funzionalità che arricchisce enormemente l'usabilità delle applicazioni è contenuta in sole 60 righe di codice.

ESEMPIO DI GUI: LAVAGNA

Ultimo esempio proposto per le interfacce è la lavagna. Elemento un po' controverso (quante volte nella vita si usa una lavagna elettronica?) è spesso presente nelle applicazioni, soprattutto quelle multimediali. Sicuramente è di grande effetto scenico.
La **figura 10** ne mostra un esempio.

Figura 10: Non è certo un disegno artistico, ma la nostra lavagna ci consente di dare libero sfogo a tutto il nostro estro

NOTA

PYTHON

Per chi volesse conoscere tutto su Python può far riferimento al sito: http://www.python.it/ Appena si entra, ci sono riferimenti all'utilizzo che viene fatto di Python, come è avvenuto, ad esempio, per il sito di Google.

```
# File: scribble.py

def __init__ (self):
        Panel.__init__(self, BL())
        self.add(Button('Clear',
                        actionPerformed=self.doClear)
, BL.SOUTH)
        self.mouseDragged = self.doDrag
        self.mousePressed = self.doPress

    def doDrag (self, event):
        g = self.graphics
        g.color = Color.black
        lx, ly = self.__last
        x = event.x; y = event.y
        g.drawLine(lx, ly, x, y) # disegna la linea
        self.__last = x, y

    def doPress (self, event):
        self.__last = event.x, event.y
```

Il codice mostra come memorizzare i movimenti del mouse e quando disegnare le linee. Con questi esempi si conclude la parte delle interfacce grafiche per lasciare il posto alla gestione dei dati.

PERSISTENZA DEI DATI

Questa seconda parte tratta principalmente l'utilizzo di Jython per memorizzare o interagire con i dati.
Questo approccio risulta particolarmente utile per accedere direttamente ai dati, per modificarli o per verificarne la consistenza. Fino ad ora si è cercato di non nominare la parola database perché sarebbe limitante nei confronti delle potenzialità che offre il linguaggio. Infatti, Jython, permette la gestione dei dati sia con i classici database che su file.
Il prossimo esempio parla proprio di questa seconda possibilità, mentre gli altri sfrutteranno l'accesso ai db. Vedremo come anche in questo caso esistano soluzioni sufficientemente eleganti e semplici da realizzare

MEMORIZZAZIONE SU FILE

Se l'applicazione che dobbiamo fare deve tenere in memoria pochi dati e se non servono ricerche particolarmente complesse, si può pensare di utilizzare i file DBM per scrivere i valori di interesse. Usare questi file è molto semplice con *Jython* e il prossimo esempio lo dimostra.

```
# File: dbm.py
import dumbdbm
dbm = dumbdbm.open("dbmtest")
dbm['Value 1'] = 'A'
dbm['Value 2'] = 'B'
dbm['Value 3'] = 'B'
for key in dbm.keys():
    print key, ":", dbm[key]
```

Con poche istruzioni si memorizzano su file i valori. Il file DBM è simile a un dizionario che contiene le coppie chiave-valore.
Fisicamente, si può verificare che, nel file system, sono creati due file uno *.dir* e uno *.dat* con lo stesso nome del file dichiarato nel codice.
Quindi, per operazioni elementari è assai efficace questo metodo, mentre per operazioni più complesse ci si deve appoggiare ai database. In realtà in questo primo caso abbiamo notato quanto trasparente allo sviluppatore sia l'intera gestione delle operazoni di scrittura e lettura dei dati sull'hard disk.

GESTIAMO I DATABASE

Più frequentemente le applicazioni memorizzano i dati su database e *Jython* si può connettere ai db usando *Java*. Negli esempi seguenti si userà *MySQL* e la connessione *jdbc:odbc* come ponte per accedervi. Ovviamente queste scelte si possono adattare ai singoli casi, cambiando il database o l'url di connessione. Nel nostro caso, scegliere questo driver vuol dire rendere il codice facilmente utilizzabile anche per altri database. Il primo esempio è davvero minimale per consentire di capire gli aspetti più importanti di una connessione.

```
# File: db.py

import com.ziclix.python.sql as sql
// url di connessione
dburl, user, pw, drv = ("jdbc:odbc:iop","root",
"password","sun.jdbc.odbc.JdbcOdbcDriver")
// esecuzione query
db = sql.zxJDBC.connect(dburl, user, pw, drv)
cursor = db.cursor()
cursor.execute("SELECT * FROM tab")
// stampa dei risultati
for item in cursor.fetchall():
    print item
```

Veramente molto semplice per chi mastica un po' di connessioni. Si importa la classe che serve per gestire il collegamento al db, si configura l'url di connessione in accordo con i parametri impostati, si esegue la query e si stampa il risultato. Da notare come tutto ciò sia possibile grazie a una classe specifica di Jython. Questo file serve solo per riscaldarsi e per verificare che le connessioni funziono. Gli altri esempi scendono più nei dettagli e permettono operazioni più elaborate.

SCANSIONE DEI RISULTATI

Uno degli scenari tipici, quando ci si collega a un db, è di dover prendere i dati restituiti da una query. Nell'esempio precedete, si è visto l'utilizzo di *cursor* che serve a scandire gli elementi grazie al metodo *fetchall()*. Per chi viene, invece, dal mondo *Java*, si può trovare più a suo agio con *ResultSet*.

```
# File: resultSet.py
stmt = dbconn.__findattr__
                ("__connection__").createStatement()
rs = stmt.executeQuery("SELECT * FROM tab")
while rs.next():
          print rs.getString(1)
```

A parte gli elementi in comune con gli altri esempi, che non vengono riportati, le altre istruzioni servono per popolare un *ResultSet* che viene poi scandito. A questo punto spetta a voi scegliere il modo migliore per connettersi al db.

ESEMPIO DATABASE LEGGIAMO I METADATI

Una volta padroni delle connessioni, si può generalizzare il tutto. Unendo le capacità grafiche e di connessione al db, si può creare una interfaccia grafica che permetta di scegliere a quale db collegarsi e che ottiene le principali informazioni relative al db stesso, i metadati. Come si può intuire dalla descrizione dell'esempio, è possibile dividere il tutto in due file: il primo che contiene l'interfaccia grafica e che si occupa della connessione e il secondo che reperisce i metadati e li formatta in maniera opportuna. Il codice del primo file è riportato di seguito.

```
# File: DBLoginDialog.py
...
class DBLoginDialog(awt.Dialog):
...
   def login(self, e):
        db = self.DBMS.selectedItem
        url = "jdbc:odbc:" + self.Database.text

        try:
           dbcon = sqlZX.zxJDBC.connect(url,
                                     self.User.text,
              self.Password.text,
                   "sun.jdbc.odbc.JdbcOdbcDriver")
           self.dispose()
           self.client(dbcon)
        except sql.SQLException:
           self.showError("Unable to connect to
                                        database")
```

È davvero solo l'applicazione di quanto visto fino adesso. A parte tutta la parte grafica, che ma-

Figura 11: GUI per selezionare il db

```
Method                                          MySQL
-------------------------------------------------------------
getCatalogTerm                                  database
getMaxBinaryLiteralLength                       0
getMaxCatalogNameLength                         64
getMaxCharLiteralLength                         0
getMaxColumnsInGroupBy                          0
getMaxColumnsInIndex                            32
getMaxColumnsInOrderBy                          0
getMaxColumnsInSelect                           0
getMaxColumnsInTable                            0
getMaxConnections                               0
getMaxCursorNameLength                          18
getMaxIndexLength                               500
getMaxProcedureNameLength                       0
getMaxRowSize                                   0
getMaxStatementLength                           8192
getMaxStatements                                0
getMaxTablesInSelect                            31
getMaxUserNameLength                            16
supportsANSI92EntryLevelSQL                     1
supportsBatchUpdates                            1
supportsCatalogsInDataManipulation              1
supportsCoreSQLGrammar                          1
supportsDataManipulationTransactionsOnly        0
supportsDifferentTableCorrelationNames          1
supportsFullOuterJoins                          0
supportsGroupByUnrelated                        1
```

Figura 12: Metadati del db

gari può essere adattata in base ai propri gusti, l'obiettivo è quello di presentare una schermata come quella in **figura 11**.
E di leggere i valori per effettuare la connessione al db. I dati che l'utente scrive a video sono letti, nel codice sopra e usati per creare l'url di connessione. In caso di problemi, viene lanciata un'eccezione. Se invece la connessione va a buon fine, interviene il secondo file che ha il compito di interrogare il db e restituirne i metadati.

```
# File: metaData.py
...
def gatherMetaInfo(self, dbconn):

        metaData =
  dbconn.__findattr__("__connection__").getMetaData()
        dbname = metaData.databaseProductName
        for cmd in self.getCommands():
            value = ""
            try:
                exec("value = metaData.%s()" % cmd)
            except:
                value = "Test failed"
            ...

    def getCommands(self):
        return ['getCatalogTerm',
                    'getMaxBinaryLiteralLength',
                'getMaxCatalogNameLength',
                    'getMaxCharLiteralLength',
                ...
```

Scorrendo il codice, ci sono due metodi che servono per prendere i metadati. Il primo prende tutte le informazioni, in generale. Il secondo specifica esattamente quali elementi del db andare a considerare: un esempio degli elementi considerati è riportato in **figura 12**.
Un esempio del genere può essere una buona base per realizzare un'interfaccia grafica che tenga sotto controllo tutti i db che abbiamo, lanciando query specifiche che mostrano l'incremento dei dati oppure quanti utenti sono connessi al db.

DATABASE: STRATIFICAZIONE

In questi esempi siamo passati da poche istruzioni a più file, aumentando la complessità del codice. Per cercare di rendere il progetto semplice, si può stratificare ulteriormente il lavoro inserendo le informazioni della connessione al db in un file a parte. Generalmente si è soliti chiamare questo file dbexts.ini che va messo nel path in cui lanciamo il progetto.

```
# File: dbexts.ini
[JDBC]
NAME=MYSQLIOP
URL=JDBC:ODBC:IOP
USER=ROOT
PWD=PASSWORD
```

Le righe sopra specificano i parametri di connessione, mentre le istruzioni seguenti servono per richiamarle.

```
# File: dbextCall.py
from dbexts import dbexts
mysqlcon = dbexts("mysqliop", "dbexts.ini")
```

Il grande vantaggio di questo approccio è di mettere in unico punto tutte le informazioni relative al db, senza doverle replicare.

CONCLUSIONI

In questo articolo è stato presentato Jython, la fusione tra Python e Java, attraverso una serie di esempi che ne illustrano le possibilità e i campi di applicazione. Con Jython, infatti, diventa più semplice realizzare interfacce grafiche che rendono più piacevole il lavoro dell'utente e collegamenti ai vari database, magari per effettuare dei test o dei controlli. Inoltre, i diversi esempi, permettono di avere una ampia gamma di scelta per poter poi personalizzare le proprie applicazioni.

Cristiano Bellucci

L'SMS CHE COMANDA IL TUO CELLULARE

IMPARIAMO COME CONFIGURARE UN CELLULARE, SINCRONIZZARE LA RUBRICA DEGLI APPUNTAMENTI, AGGIUNGERE UN CONTATTO E MOLTO ALTRO ANCORA SEMPLICEMENTE INVIANDOGLI UN BREVE MESSAGGIO DI TESTO DA REMOTO

In questo articolo analizzeremo alcune nuove API esposte in Windows Mobile 5 e creeremo una infrastruttura che ci consentirà di comandare il nostro dispositivo (sia Smartphone sia Pocket PC) attraverso l'invio di messaggi SMS formattati opportunamente. Gli utilizzi di questo genere di applicazione sono svariati. Ad esempio potremmo inviare un SMS ad un cellulare per configurarlo secondo certe specifiche di un provider. Al termine dell'articolo vedremo diversi esempi d'uso.

LA STRUTTURA DEI COMANDI

Prima di addentrarci nello sviluppo del codice della nostra applicazione, dobbiamo risolvere una questione di carattere teorico, ovvero dobbiamo decidere la struttura dei messaggi che vogliamo utilizzare come comandi. La scelta di una buona struttura del comando è fondamentale per rendere più agevole la relativa decodifica ed esecuzione. Nel presente articolo, la struttura di un messaggio di comando sarà:

```
<CMD:><Comando><Parametri>
```

Dove *<CMD:>* rappresenta la parola chiave che indica che il messaggio è in realtà un comando; *<Comando>* rappresenta una parola appartenente alla grammatica dei comandi che andremo a definire più avanti; *<Parametri>* rappresenta "il resto" del messaggio che verrà (o meglio potrà) essere utilizzato dalla nostra applicazione. Un esempio di comando che rispetta la struttura definita è:

```
CMD:CONFIGURA http://mio.dominio.it/smartConfigure.xml
```

La sua interpretazione è:
"esegui un comando di configurazione del dispositivo scaricando il relativo file dall'indirizzo http://mio.dominio.it/smartConfigure.xml".

INFRASTRUTTURA DI BASE

Come ambiente di sviluppo utilizzeremo Microsoft Visual Studio 2005; per iniziare dobbiamo creare un nuovo progetto. Come mostrato in **Figura 1**, dalla lista dei tipi di progetto, sceglieremo, ad esempio, Windows Mobile 5.0 Pocket PC, e tra i tipi di template *Device Application* (1.0) che ci consentirà di sviluppare una applicazione basata sulla versione 1.0 del .Net Framework. Questa scelta ci permette di utilizzare comunque tutte le funzionalità di cui abbiamo bisogno senza però avere la necessità di installare nessun altro componente nel dispositivo (nei file a supporto è possibile trovare la soluzione utilizzata nel corso dell'articolo).

***Fig. 1:** Creiamo un nuovo progetto basato su .Net Framework 1.0.*

Il primo passo da compiere nello sviluppo della nostra soluzione, è quello di aggiungere un riferimento alle dll che contengono le funzionalità che utilizzeremo nel seguito della trattazione. Dal pannello della soluzione facciamo clic con il tasto destro sulla voce *Riferimenti*, quindi facciamo clic con il tasto sinistro su *Aggiungi Riferimento*. Il dialogo che ci appare mostra la lista delle dll disponibili.
Le nuove classi che andremo ad usare sono contenute in *Microsoft.WindowsMobile.Configuration* ed in *Microsoft.WindowsMobile.PocketOutlook*, quindi selezioniamo le voci dalla lista e premiamo *Ok* (si veda **Figura 2**). Una volta fissato il riferimento alle dll, nel codice aggiungiamo delle clausole *using* in modo da

Fig. 2: Aggiungiamo i riferimenti alle dll che contengono le nuove classi.

poter utilizzare le classi dichiarando semplicemente il loro nome senza dover ogni volta specificare il namespace di appartenenza.

```
using Microsoft.WindowsMobile.Configuration;
using Microsoft.WindowsMobile.PocketOutlook;
using Microsoft.WindowsMobile.PocketOutlook
                                    .MessageInterception;
```

Per inizializzare il processo di intercettazione, definiamo un metodo privato che viene invocato all'avvio dell'applicazione *(InitializeInterceptor())*.
Nella prima parte del metodo carichiamo un oggetto di tipo *XmlDocument* con un frammento XML che contiene alcuni valori di configurazione che utilizzeremo in seguito. Il documento XML di configurazione ha lo stesso nome dell'eseguibile ed è memorizzato nella stessa cartella.

```
// Creiamo il documento che conterrà i valori di
// configurazione
oXDoc = new XmlDocument();
// Individuiamo il percorso del file di configurazione
fullyQualifiedName = (System.Reflection.Assembly.
            GetExecutingAssembly().GetModules())[0].
                                  FullyQualifiedName;
string configPath = fullyQualifiedName.Replace(
                                       ".exe", ".xml");
try
{  // Carichiamo il documento con la configurazione
   oXDoc.Load(configPath); }
catch (XmlException xExc)
{ lblError.Text = xExc.Message; }
```

L'infrastruttura di intercettazione si basa sulla classe *MessageInterceptor;* il costruttore che utilizzeremo prende in input due parametri. Il primo indica cosa fare del messaggio intercettato; le azioni possibili sono due e vengono definite attraverso l'enumeratore *InterceptionAction*. Se specifichiamo come valore *Notify,* il messaggio intercettato viene notificato all'utente e conservato come qualsiasi altro; se specifichiamo, invece, il valore *NotifyAndDelete,* il messaggio non verrà notificato e nemmeno memorizzato. In questo secondo scenario il dispositivo reagirà in modo attivo ai comandi senza interferire nelle attività dell'utente; se da una parte questo comportamento è ideale per comandi di configurazione o per interagire con *Pocket Outlook,* dall'altra è necessario che l'utente sia consapevole di avere un dispositivo "attivo" in modo da non disorientarsi al vedere nuovi appuntamenti o nuove configurazioni attivate in modo "silente". Nell'esempio che utilizzeremo nel corso della trattazione si è scelto di notificare i messaggi in modo da avere anche una traccia del comportamento del sistema.

```
// Definiamo la sorte dei messaggi di comando che NON
// saranno cancellati
InterceptionAction oIAction = InterceptionAction.Notify;
// Creiamo un nuovo oggetto di tipo MessageInterceptor
oSmsInterceptor = new MessageInterceptor(
                                    oIAction, true);
```

Al fine di ottimizzare la procedura di intercettazione e, soprattutto, al fine di intercettare solo quei messaggi che corrispondono a comandi, è possibile definire una "condizione", il cui verificarsi comporta il riconoscimento del messaggio come comando.
L'infrastruttura messa a disposizione da Windows Mobile 5, ci consente innanzitutto di decidere in che "parte" del messaggio andare a verificare la condizione; attraverso l'enumeratore *MessageProperty* possiamo selezionare ad esempio il corpo del messaggio oppure il mandante. Nel caso sotto esame abbiamo definito che il comando è contenuto nel corpo del messaggio.

```
MessageProperty oMProperty = MessageProperty.Body;
```

Dopo aver individuato in che parte del messaggio effettuare la verifica della condizione, dobbiamo definire che tipo verifica intendiamo fare; attraverso l'enumeratore *MessagePropertyComparisonType* possiamo decidere se la condizione è verificata quando la proprietà scelta (il corpo del messaggio) "comincia per" oppure "finisce per" oppure "contiene" una data stringa. Nel nostro caso decidiamo che il messaggio è un comando quando "comincia per" una stringa che definiremo in fase di definizione della condizione.

```
MessagePropertyComparisonType oMPComparition =
            MessagePropertyComparisonType.StartsWith;
```

Definite le diverse parti della condizione possiamo valorizzare la proprietà *MessageCondition* del nostro oggetto *MessageInterceptor* al fine di consentire la corretta intercettazione dei messaggi:

SUL WEB

Una estesa trattazione ("da una prospettiva da sviluppatore") su tutte le novità del nuovo sistema operativo Windows Mobile 5, è reperibile nella Library MSDN nell'articolo:
• WHAT'S NEW FOR DEVELOPERS IN WINDOWS MOBILE 5.0
a partire dall'indirizzo:
http://msdn2.microsoft.com/en-us/library/ms839548.aspx

```
oSmsInterceptor.MessageCondition =
                    new MessageCondition(oMProperty,
                      oMPComparition, "CMD:", true);
```

Da quanto detto in precedenza, la condizione va letta in questo modo: "Intercetta tutti quei messaggi il cui corpo inizia per CMD:"; l'ultimo parametro, valorizzato a true ci consente di imporre che la condizione sia anche sensibile alle maiuscole ("case sensitive"). Come ultima istruzione, dobbiamo aggiungere un gestore dell'evento che reagisca all'intercettazione di un messaggio di comando; ciò si attua valorizzando la proprietà *MessageReceived* del nostro oggetto *MessageInterceptor*.

```
oSmsInterceptor.MessageReceived += new
                     MessageInterceptorEventHandler(
                    smsInterceptor_MessageReceived);
```

Di seguito il codice completo del metodo *InitializeInterceptor()*.

```
private void InitializeInterceptor()
{ // Creiamo il documento che conterrà i valori di
  // configurazione
   oXDoc = new XmlDocument();
   // Individuiamo il percorso del file di configurazione
   fullyQualifiedName = (System.Reflection.Assembly.
   GetExecutingAssembly().GetModules())[0]
                                  .FullyQualifiedName;
   string configPath = fullyQualifiedName.Replace(
                                      ".exe", ".xml");
   try
   {  // Carichiamo il documento con la configurazione
      oXDoc.Load(configPath); }
   catch (XmlException xExc)
   { lblError.Text = xExc.Message;}
   try
   { // Definiamo la sorte dei messaggi di comando che
     // NON saranno cancellati
      InterceptionAction oIAction = InterceptionAction.Notify;
      // Creiamo un nuovo oggetto di tipo MessageInterceptor
      oSmsInterceptor = new MessageInterceptor(
                                     oIAction, true);
      // Definiamo su che campo deve essere verificata la
      // condizione
      MessageProperty oMProperty = MessageProperty.Body;
      // Definiamo il tipo di confronto della condizione
      MessagePropertyComparisonType oMPComparition =
      MessagePropertyComparisonType.StartsWith;
      // Definiamo la condizione affinchè un messaggio sia
      // riconosciuto come comando
     oSmsInterceptor.MessageCondition = new
          MessageCondition(oMProperty, oMPComparition,
                                       "CMD:", true);
     // Aggiungiamo un gestore dei messaggi riconosciuti
     // come comandi
     oSmsInterceptor.MessageReceived +=
        new MessageInterceptorEventHandler(
                  smsInterceptor_MessageReceived); }
   catch (Exception exc)
   { lblError.Text += exc.Message;}
}
```

IL PARSER DEI COMANDI

Una volta definita l'infrastruttura di base della nostra applicazione, dobbiamo implementare il gestore dell'evento che reagisce all'intercettazione di un messaggio di comando. Come possiamo immaginare, questo metodo non è nient'altro che un parser (o analizzatore sintattico) dei comandi che, a partire dalla grammatica definita all'inizio della trattazione, deve interpretare ed eseguire il comando contenuto nel messaggio. La sua implementazione comincia con la cattura del messaggio ricevuto:

```
// Istanziamo un oggetto "SMS" con il messaggio intercettato
SmsMessage sms = (SmsMessage)e.Message;
```

Quindi dobbiamo isolare il corpo del messaggio, eliminare la parte iniziale che contiene solo la parola chiave (*"CMD:"*), infine isolare il comando da eseguire.

```
// Isoliamo il contenuto del messaggio
string smsBody = sms.Body;
// Eliminiamo la parte iniziale del messaggio ("CMD:")
string message = smsBody.Substring(4);
int commandLen = message.IndexOf(' ');
// Isoliamo il comando specifico (<Comando><Parametri>)
string command = message.Substring(0, commandLen);
```

L'ultima parte del metodo è un semplice "switch" che, sulla base del comando individuato, esegue il relativo codice di attuazione.

```
private void smsInterceptor_MessageReceived(
          object sender, MessageInterceptorEventArgs e)
{  // Istanziamo un oggetto "SMS" con il messaggio
   // intercettato
   SmsMessage sms = (SmsMessage)e.Message;
   // Isoliamo il contenuto del messaggio
   string smsBody = sms.Body;
   // Eliminiamo la parte iniziale del messaggio ("CMD:")
   string message = smsBody.Substring(4);
   int commandLen = message.IndexOf(' ');
   // Isoliamo il comando specifico (<Comando><Parametri>)
   string command = message.Substring(0, commandLen);
   // Individuiamo il comando da eseguire
   switch (command)
   {  case "CIRCOLA":
         // Gestione Messaggi Circolari
         sendCircularMessage(message);
         break;
      case "CONFIGURA":
```

```
            // Gestione Configurazione Remota
            getConfiguration(message.Substring(9));
            break;
        case "OUTLOOK":
            // Gestione Pocket Outlook
            manageOutlook(message.Substring(7));
            break; }
}
```

Nel seguito della trattazione verranno definiti tre esempi che dimostrano come mettere in pratica l'intercettazione dei messaggi SMS. Vedremo come sia possibile far sì che il dispositivo venga configurato remotamente attraverso lo scarico di un file di configurazione; inoltre vedremo come implementare una struttura di messaggistica "a catena"; infine vedremo come interagire con il nuovo Pocket Outlook.

CONFIGURAZIONE REMOTA

L'intercettazione di messaggi SMS, può esserci di grande aiuto nella configurazione di dispositivi remoti. Per implementare questa soluzione, utilizzeremo il provisioning via XML, ovvero la capacità di configurare i dispositivi mobili basati su Windows Mobile tramite documenti XML. In un precedente articolo è stato dettagliatamente trattato l'argomento del provisioning via XML, quindi, in questa sede, ci basterà sapere che, al fine di consentire la configurazione remota, dobbiamo attivare un server web in modo che possa ospitare il documento XML di configurazione e far sì che questo sia raggiungibile dal nostro dispositivo. Nell'esempio che stiamo considerando, nella nostra macchina di sviluppo sarà attivato anche Internet Information Server in cui definiremo una cartella virtuale al cui interno andremo a memorizzare il nostro documento di configurazione. La struttura del messaggio SMS che dovremo inviare al dispositivo, ricordando anche la grammatica definita in precedenza, sarà:

```
CMD:CONFIGURA <URL Documento di configurazione>
```

Dove <*URL Documento di configurazione*> rappresenta l'indirizzo a cui dovremo connetterci per scaricare il documento di configurazione.
Se analizziamo il codice di esecuzione del comando *(getConfiguration())*, come prima cosa dobbiamo effettuare una richiesta al server web il cui indirizzo ci viene passato in input; utilizziamo il metodo *Create()* della classe *WebRequest* passando come parametro l'indirizzo del documento di configurazione.

```
// Effettuiamo una richiesta al server il cui indirizzo
// ci viene passato in input
WebRequest oRequest = WebRequest.Create(requestURL);
```

Dopodiché dobbiamo analizzare la risposta del server attraverso un oggetto di tipo *WebResponse* che creiamo invocando il metodo *GetResponse()* dell'oggetto di tipo *WebRequest* appena creato.

```
// Gestiamo la risposta del server
WebResponse oResponse = oRequest.GetResponse();
```

Per leggere i dati inviati dal server in risposta alla nostra richiesta, prima dobbiamo "canalizzarli" in un flusso (*Stream*), quindi li andremo a leggere con un oggetto di tipo *StreamReader* (dopo aver definito opportunamente il tipo di codifica da utilizzare).

```
// Canalizziamo la risposta del server su di uno stream
Stream oResponseStream = oResponse.
                                GetResponseStream();
// Definiamo la giusta codifica per la lettura dello stream
Encoding oEncode = System.Text.Encoding.
                                GetEncoding("utf-8");
// Creiamo un "lettore" dello stream
StreamReader oStreamReader = new
            StreamReader(oResponseStream, oEncode);
```

Per poter utilizzare i dati li "scarichiamo" su di una stringa utilizzando il metodo *ReadToEnd()* dell'oggetto di tipo *StreamReader*.

```
// Scarichiamo il contenuto dello stream in una stringa
sConfigDoc = oStreamReader.ReadToEnd();
```

A questo punto, se tutte le operazioni sono andate a buon fine, la stringa *sConfigDoc* contiene esattamente il documento XML che dobbiamo utilizzare per configurare il dispositivo. Dopo aver opportunamente chiuso tutti i flussi, possiamo caricare il documento ricevuto, in un oggetto di tipo *XmlDocument* invocando il metodo *LoadXml()*.

```
// Creiamo un nuovo oggetto "Documento XML"
XmlDocument xConfigDoc = new XmlDocument();
// Istanziamo il nuovo oggetto con il documento
// da utilizzare per la configurazione
xConfigDoc.LoadXml(sConfigDoc);
```

Come ultima operazione non ci resta che invocare il *Configuration Manager* che si occupa di applicare le configurazioni che gli vengono fornite attraverso il documento XML di configurazione. Il *Configuration Manager* è implementato dalla classe *ConfigurationManager* che è anch'essa parte delle nuove funzionalità messe a disposizione da Windows Mobile 5; per poter utilizzare tale classe abbiamo già aggiunto un riferimento all'assembly che la contiene ovvero *Microsoft.WindowsMobile.Configuration* ed abbiamo anche aggiunto una clausola *using* all'omonimo namespace in modo da poter utilizzare direttamente la classe senza dover specificare per intero la struttura del

NOTA

GESTIRE GLI ERRORI

Durante il debug delle nostre applicazioni per dispositivi mobili, può capitare inevitabilmente di incappare in errori non "trappati" correttamente che danno luogo ad eccezioni. Se ci capita di vedere lo strano messaggio "Could not find resource assembly" non dobbiamo allarmarci; per motivi di spazio (siamo in dispositivi mobili ancora abbastanza limitati) i file che contengono i messaggi di errore "parlanti" non sempre vengono caricati nel dispositivo. Per ovviare a tale problema è semplicemente necessario copiare ed installare nel dispositivo il file System_SR_IT.cab che contiene i suddetti messaggi (dopo l'installazione è necessario il restart del dispositivo). Un interessante articolo sull'argomento è reperibile all'indirizzo:
http://blogs.msdn.com/netcfteam/archive/2004/08/06/210232.aspx

namespace. L'applicazione della configurazione avviene invocando il metodo statico *ProcessConfiguration* della classe *ConfigurationManager;* questo, prende in input proprio l'oggetto *XmlDocument* che abbiamo appena creato *(xConfigDoc);* come secondo parametro di input prende un valore booleano che, se impostato a true, comporta la restituzione del documento stesso con l'aggiunta di tag opportuni per evidenziare gli eventuali errori riscontrati nel processo.

```
// Inviamo il documento di configurazione al Configuration
// Manager
ConfigurationManager.ProcessConfiguration(
                                    xConfigDoc, false);
```

In breve, quindi, la configurazione del nostro dispositivo (che potrebbe essere anche molto complessa) si attua con una sola linea di codice. Questa precisazione è importante per sottolineare ancora una volta la potenza delle primitive messe a disposizione da Windows Mobile 5. Ovviamente, dato che praticamente ognuna delle istruzioni descritte può causare un errore (ad esempio l'indirizzo può essere errato oppure può non essere raggiungibile), è buona norma racchiudere il codice appena visto in una istruzione *try* che ci consente di "trappare" eventuali errori e quindi mostrare un messaggio intelligibile all'utente (che magari se ne farà ben poco ma quantomeno potrà comunicarcelo consentendoci di correggere o almeno spiegare il malfunzionamento). Di seguito il codice di esecuzione del comando *getConfiguration()*.

```
private void getConfiguration(string requestURL) {
  string sConfigDoc = "";
  try {
    // Effettuiamo una richiesta al server il cui indirizzo
    // ci viene passato in input
    WebRequest oRequest = WebRequest.Create(
                                        requestURL);
    // Gestiamo la risposta del server
    WebResponse oResponse = oRequest.GetResponse();
    // Canalizziamo la risposta del server su di uno stream
    Stream oResponseStream = oResponse.
                                 GetResponseStream();
    // Definiamo la giusta codifica per la lettura dello stream
    Encoding oEncode = System.Text.Encoding.
                                 GetEncoding("utf-8");
    // Creiamo un "lettore" dello stream
    StreamReader oStreamReader = new StreamReader(
                          oResponseStream, oEncode);
    // Scarichiamo il contenuto dello stream in una stringa
    sConfigDoc = oStreamReader.ReadToEnd();
    // Chiudiamo tutti i flussi
    oStreamReader.Close();
    oResponseStream.Close();
    oResponse.Close();
    // Creiamo un nuovo oggetto "Documento XML"
    XmlDocument xConfigDoc = new XmlDocument();
```

```
    // Istanziamo il nuovo oggetto con il documento
    // da utilizzare per la configurazione
    xConfigDoc.LoadXml(sConfigDoc);
    // Inviamo il documento di configurazione al
    // Configuration Manager
    ConfigurationManager.ProcessConfiguration(
                                 xConfigDoc, false); }
  catch (Exception ex) {
    lblError.Text = ex.Message;}
}
```

Per verificare il comando di configurazione remota utilizziamo *Microsoft Device Emulator* che emula dispositivi basati su Windows Mobile 5. Utilizzeremo altresì il numero 14250010001 che consente di inviare messaggi all'emulatore stesso. La **Figura 3** mostra uno storyboard in cui sono raffigurate le fasi attraverso cui possiamo verificare il funzionamento della nostra applicazione quando viene inviato un messaggio di comando di configurazione remota. Nel test utilizziamo il file di configurazione *smartConfigure.xml* che possiamo trovare nei file di supporto e che effettua l'aggiunta di due preferiti a Pocket Internet Explorer. Si presuppone che sulla macchina di test ci sia attivo Microsoft Internet Information Server con una directory virtuale dove viene ospitato il file di configurazione. Il messaggio che inviamo al dispositivo è il seguente:

```
CMD:CONFIGURA http://10.6.97.127
                  /XMLProvisioning/smartConfigure.xml
```

Se proviamo a descrivere i diversi passi:

A. Innanzitutto verifichiamo lo stato attuale dei preferiti di Pocket Internet Explorer;

B. Compiliamo il messaggio che indica alla applicazioni di scaricare il file di configurazione da un dato indirizzo (per effettuare il test sulla propria macchina è necessario modificare l'indirizzo IP con quello della propria macchina); alla fine della compilazione premiamo *Send* per inviare il

***Fig. 3:** Configurazione remota tramite invio di messaggio SMS.*

NOTA

DOVE TROVARE L'EMULATORE!

Microsoft Device Emulator è compreso in Microsoft Visual Studio 2005 oppure è possibile scaricarlo dal sito Microsoft a partire dall'indirizzo http://msdn.microsoft.com/mobility/windowsmobile/downloads/default.aspx dove è possibile scaricare anche l'SDK di Windows Mobile 5.0 per Pocket PC e per Smartphone; questo componente aggiunge gli emulatori per la piattaforma Windows Mobile 5.0 al Device Emulator.

messaggio;

C. Il messaggio è ricevuto correttamente; dato che l'applicazione è configurata per notificare anche i messaggi di comando, questo viene mostrato come qualsiasi altro messaggio;

D. Verifichiamo che i preferiti di Pocket Internet Explorer sono variati così come da file di configurazione.

INTERAZIONE CON MICROSOFT POCKET OUTLOOK

Le grandi organizzazioni che possono permettersi una infrastruttura di messaggistica basata su Microsoft Exchange, possono sfruttare le possibilità di sincronizzazione via etere dei dispositivi mobili. Con tale caratteristica, tutti i dispositivi mobili aziendali hanno la capacità di sincronizzare i contatti, l'agenda, le attività e la posta elettronica con il server Exchange. Indubbiamente l'investimento necessario per installare una infrastruttura basata su Exchange non è concepibile per le piccole organizzazioni e tanto meno per singoli professionisti. Di seguito vedremo come le nuove primitive messe a disposizione da Windows Mobile 5 ci consentano di sincronizzare alcune informazioni via etere riproducendo, in parte, il meccanismo implementato con Microsoft Exchange.
Innanzitutto dobbiamo stabilire la grammatica del nuovo comando. In questo caso possiamo immaginare di strutturare il comando su più livelli nel senso che un primo livello ci consentirà di capire che il messaggio di comando intende interagire con Microsoft Pocket Outlook; un secondo livello ci consentirà di specificare più dettagliatamente che tipo di interazione si intende eseguire (gestione di un appuntamento, piuttosto che di un contatto); un terzo livello, infine, ci consentirà di specificare se l'operazione da fare è una aggiunta o una cancellazione.
In questo articolo vedremo come gestire le interazioni con gli appuntamenti, i contatti e le attività.
La struttura del messaggio sarà:

```
CMD:OUTLOOK <Destinazione><AddOrDel><Parametri>
```

Dove *<Destinazione>* può assumere uno tra i seguenti valori: *APP* per gestire gli appuntamenti; *CON* per gestire i contatti; *TSK* per gestire le attività. *<AddOrDel>* può assumere uno tra i seguenti valori: *ADD* per effettuare l'aggiunta di un elemento; *DEL* per effettuare una cancellazione. *<Parametri>* sarà una lista separata dal carattere *"pipe" (|)*, contenente le informazioni necessarie per gestire l'informazione relativa. Se analizziamo il codice di esecuzione del comando *(manageOutlook())*, come prima cosa dobbiamo stabilire una connessione con Pocket Outlook ovvero apriamo una sessione:

```
// Apriamo una sessione con Pocket Outlook
OutlookSession oOSession = new OutlookSession()
```

Quindi dobbiamo analizzare il contenuto del comando, cominciando ad isolare il secondo livello che ci indica l'ambito su cui agire (appuntamenti, contatti o attività).

```
// Isoliamo il comando di II livello
string subCommand = command.Substring(1, 3);
```

Infine immagazziniamo i parametri (compreso il tipo di comando da eseguire) in un array.

```
// Isoliamo i paramentri in un Array
string[] arrParameters = command.Substring(5)
                        .Split(new char[] { '|' });
```

Analizzando, attraverso uno statement switch, il contenuto del comando di secondo livello possiamo attivare il processo nel relativo ambito.

```
switch (subCommand) {
  case "APP":
    // Gestione Appuntamenti
    manageAppointment(arrParameters, oOSession);
    break;
  case "CON":
    // Gestione Contatti
    manageContact(arrParameters, oOSession);
    break;
  case "TSK":
    // Gestione Attività
    manageTask(arrParameters, oOSession);
    break;
}
```

Per facilitare la gestione andremo ad incapsulare in altrettanti metodi la gestione degli appuntamenti, dei contatti e delle attività.

GESTIONE DEGLI APPUNTAMENTI

Attraverso il metodo *manageAppointment()* gestiamo gli appuntamenti che sono implementati dalla classe *Appointment* contenuta nel namespace *Microsoft.WindowsMobile.PocketOutlook*. Creiamo quindi un oggetto di tipo *Appointment*:

```
// Creiamo il nuovo oggetto "Appuntamento"
Appointment oAppointment = new Appointment();
```

Di seguito dobbiamo valorizzare le diverse proprietà dell'oggetto con i valori dei parametri che ci arrivano in input. La classe *Appointment* è veramente ricca ed esistono numerose proprietà che possono essere va-

lorizzate. Nel contesto attuale ci basterà definire un appuntamento specificando solamente l'argomento *(Subject)*, la data di inizio *(Start)* e la data di fine *(End)*, che sono gli elementi minimali per la sua definizione.

```
// Istanziamo i valori
oAppointment.Subject = arrParameters[1];
oAppointment.Start = buildDateTime(arrParameters[2]);
oAppointment.End = buildDateTime(arrParameters[3]);
```

Chiaramente, passando un numero diverso di informazioni sarà possibile gestire qualsiasi aspetto dell'appuntamento.
Possiamo notare che la proprietà Subject è di tipo stringa quindi può essere inizializzata direttamente con il valore del parametro di input; le proprietà *Start* ed *End*, invece, essendo di tipo *DateTime*, devono essere inizializzate con un valore opportunamente definito utilizzando la funzione illustrata di seguito:

```
private DateTime buildDateTime(string sDateTime) {
  // Separiamo la data dall'ora
  string[] arrDateTime = sDateTime.Split(new char[] { ' ' });
  // Isoliamo le componenti della data
  string[] arrDate = arrDateTime[0].Split(new char[] { '/' });
  // Isoliamo le componenti dell'orario
  string[] arrTime = arrDateTime[1].Split(new char[]
                                                    { ':' });
  // Creiamo un nuovo oggetto di tipo DateTime con le
  // componenti
  DateTime oDateTime = new DateTime(
    Convert.ToInt32(arrDate[2]),
    Convert.ToInt32(arrDate[1]),
    Convert.ToInt32(arrDate[0]),
    Convert.ToInt32(arrTime[0]),
    Convert.ToInt32(arrTime[1]),
    Convert.ToInt32(arrTime[2]));
  return oDateTime;
}
```

La funzione attende in input una stringa con il formato "gg/mm/aaaa hh:mm:ss"; dopo aver opportunamente isolato le varie componenti della data e dell'orario non fa altro che creare un opportuna istanza della classe *DateTime* passando le singole componenti. Tornando alla gestione degli appuntamenti, una volta definito l'oggetto di tipo *Appointment*, dobbiamo analizzare il valore del comando di terzo livello che ci indica se effettuare una aggiunta od una cancellazione. Nel primo caso dobbiamo solo aggiungere l'appuntamento appena creato alla relativa lista; utilizziamo il metodo *Add()* della collezione *Items* esposta dalla collezione *Appointments* della sessione Pocket Outlook, aperta in testa al metodo di gestione del comando e passata come parametro *(oOSession)*.

```
// Aggiungiamo il nuovo appuntamento
oOSession.Appointments.Items.Add(oAppointment);
```

Nel caso in cui dovessimo invece cancellare un appuntamento, dobbiamo utilizzare una procedura meno immediata. Tra le diverse possibilità, la più flessibile è offerta dal metodo *Restrict()* sempre della collezione Items usata in precedenza. Il metodo consente di selezionare un sottoinsieme di elementi della collezione, a partire da una stringa che definisce il criterio di selezione; la stringa deve contenere una espressione, di tipo booleano, che viene valutata per ogni elemento della collezione; i nomi delle proprietà da utilizzare nella selezione devono essere racchiusi tra parentesi quadre ([]) mentre i valori testuali da usare nei confronti, devono essere racchiusi tra doppi apici (""); è possibile combinare diverse espressioni con operatori *AND* ed *OR*. Nel caso che stiamo analizzando utilizzeremo tutte le proprietà che ci arrivano dal messaggio SMS, ovvero l'argomento *(Subject)* e le date di inizio e fine dell'appuntamento *(Start, End)*.

```
// Ricerchiamo l'appuntamento da cancellare
AppointmentCollection oACollection =
oOSession.Appointments.Items.Restrict(
                    "[Subject] = \"" + arrParameters[1] +
                     "\" AND [Start] = " + buildDateTime(
                          arrParameters[2]).ToString() +
  " AND [End] = " + buildDateTime(
                          arrParameters[3]).ToString());
```

Possiamo notare come i valori delle date NON debbano essere messi tra apici dato che non rappresentano valori stringa. È solo il caso di evidenziare che l'utilizzo di questo metodo potrebbe essere utilizzato in maniera ulteriormente flessibile per operare in modo massivo su insiemi di appuntamenti. È altresì interessante notare che, indipendentemente dalle proprietà che possono essere gestite "in effettivo", la stringa di selezione necessaria per l'individuazione di un appuntamento può essere composta solo da quelle proprietà che individuano univocamente un appuntamento; possiamo quindi affermare con tranquillità che la stringa utilizzata nel presente articolo può andar bene anche in una situazione in cui le informazioni passate via SMS siano più numerose.
Il risultato della selezione è memorizzato in una collezione di appuntamenti *(oACollection)* che dovremo scorrere per cancellare ogni elemento.

```
foreach (Appointment oApp in oACollection) {
  // Cancelliamo l'appuntamento
  oOSession.Appointments.Items.Remove(oApp);
}
break;
```

La cancellazione dell'appuntamento avviene invocando il metodo *Remove()* della solita collezione *Items*. Di seguito il codice del metodo.

```
private void manageAppointment(string[] arrParameters,
```

```
                              OutlookSession oOSession)
{
  try
  {
    // Creiamo il nuovo oggetto "Appuntamento"
    Appointment oAppointment = new Appointment();
    // Istanziamo i valori
    oAppointment.Subject = arrParameters[1];
    oAppointment.Start = buildDateTime(
                                      arrParameters[2]);
     oAppointment.End = buildDateTime(
                                      arrParameters[3]);
     // Decidiamo l'operazione da compiere
     switch (arrParameters[0]){
         case "ADD":
             // Aggiungiamo il nuovo appuntamento
             oOSession.Appointments.Items.Add(
                                         oAppointment);
             break;
         case "DEL":
             // Ricerchiamo l'appuntamento da cancellare
             AppointmentCollection oACollection =
             oOSession.Appointments.Items.Restrict(
                     "[Subject] = \"" + arrParameters[1] +
                     "\" AND [Start] = " + buildDateTime(
                         arrParameters[2]).ToString() +
                     " AND [End] = " + buildDateTime(
                         arrParameters[3]).ToString());
             foreach (Appointment oApp in oACollection)
             {
               // Cancelliamo l'appuntamento
               oOSession.Appointments.Items.Remove(oApp);
             }
             break;
     }
  }
  catch (Exception ex)
  {lblError.Text = ex.Message;}
}
```

Per verificare il funzionamento dell'applicazione relativamente alla gestione degli appuntamenti, inviamo al dispositivo il seguente messaggio:

```
CMD:OUTLOOK APP ADD|Appuntamento per discussione
      articolo|22/02/2007 10:00:00|22/02/2007 11:00:00
```

La sua interpretazione è: "Aggiungere un appuntamento per il 22 Febbraio dalle 10 alle 11 del mattino per discussione su articolo".
Se commentiamo lo storyboard di **Figura 5** che mostra l'esempio in pratica:

A) Verifichiamo che non vi siano appuntamenti nel prossimo futuro;
B) Compiliamo ed inviamo il messaggio in modo che sia fissato un appuntamento per il 22 di Febbraio;

Fig. 5: Definizione di un appuntamento tramite messaggio SMS

C) Verifichiamo che il messaggio sia arrivato correttamente;
D) Verifichiamo che l'appuntamento sia stato effettivamente aggiunto.

DISTRIBUZIONE DELL'APPLICAZIONE

A supporto del presente articolo possiamo trovare la soluzione per Visual Studio 2005 che implementa l'applicazione di intercettazione dei messaggi SMS. Per distribuire l'applicazione innanzitutto dobbiamo lanciare la "build" della soluzione che creerà l'eseguibile SMSInterceptor.exe. Dopodiché possiamo tranquillamente copiare l'eseguibile nel dispositivo, in una qualsiasi cartella, per poi lanciarlo sul dispositivo affinché venga attivata la procedura di intercettazione. In fase di distribuzione dell'applicazione è necessario ricordarsi di spostare sul dispositivo, nella stessa cartella dell'eseguibile, anche il documento di configurazione chiamato SMSInterceptor.xml che abbiamo usato nell'esempio dei messaggi circolari e che è parte della soluzione.

CONCLUSIONI

Con l'ultimo sistema operativo per dispositivi mobili, Windows Mobile 5, Microsoft ha ulteriormente allargato le funzionalità dedicate agli sviluppatori in ambito .Net Framework.
Nel presente articolo abbiamo avuto occasione di prendere conoscenza di alcuni aspetti peculiari che ci hanno consentito di creare una infrastruttura di intercettazione di messaggi SMS attraverso cui abbiamo la possibilità di "comandare" il nostro dispositivo basato su Windows Mobile 5. Negli esempi che abbiamo sviluppato, abbiamo avuto altresì modo di osservare all'opera altre nuove primitive per la gestione della configurazione remota, per l'invio di messaggi SMS e per l'interazione con Pocket Outlook. Indubbiamente, una volta eretta l'infrastruttura di intercettazione, sono praticamente infinite le possibilità di interrogazione ed interazione verso i nostri dispositivi.

Oscar Peli

L'AUTORE

Oscar Peli è .NET Solution Architect ed amministratore dei dispositivi mobili presso il Comune di Ancona. Come consulente ha sviluppato presso l'Università degli Studi di Macerata un sistema di pubblicazione di contenuti per il supporto a progetti di ricerca http://reti.unimc.it. Oscar è contattabile all'indirizzo: opeli@unimc.it.

LOCALIZZARE SERVIZI WEB CON STRUTS 2

IMPARIAMO COME REALIZZARE APPLICAZIONI MULTILINGUA UTILIZZANDO IL FRAMEWORK DI SVILUPPO PER IL PATTERN MVC PIÙ CONOSCIUTO AL MONDO. NON SOLO OTTERREMO SOFTWARE AFFIDABILE MA ANCHE FACILMENTE DISTRIBUIBILE

Molti servizi erogati via Web hanno per varie ragioni la necessità di adattare i propri contenuti alla "cultura" dell'utente che si collega al sito. Si pensi ad esempio ad un sito dedicato ai cittadini europei che fornisce informazioni sui nuovi modelli di autovettura vendute negli stati membri della EU e in quale di questi sia più conveniente acquistarle. Ovviamente c'è la necessità di fornire i contenuti, come ad esempio le descrizione degli interni degli autoveicoli, in varie lingue: per l'utente che si collega dalla Germania in tedesco, per un belga in francese. Il prezzo dell'automobile dovrà essere mostrato nella moneta utilizzata nel Paese d'origine dell'internauta, quindi tutti gli utenti provenienti dall'area euro vedranno il prezzo espresso nella nostra moneta, mentre i navigatori slovacchi preferiranno che il prezzo sia mostrato in corone slovacche. Continuando nell'esempio un italiano preferirà che i dati tecnici delle autovetture siano espressi in unità quali Km/h per la velocità, KW per le potenze. Viceversa un utente proveniente dal Regno Unito o dall'Irlanda, affezionato al sistema imperiale, preferirà leggere le stesse grandezze espresse in Mph o HP. Ecco quindi l'esempio dell'importanza della localizzazione delle applicazioni nel caso si voglia raggiungere una vasta platea.Nello scorso numero di ioProgrammo abbiamo iniziato a sviluppare un'applicazione di esempio dedicata alle agenzie immobiliari. Utilizzeremo la stessa applicazione come esempio per di localizzazione. Con molta probabilità, nella societa multietnica in cui viviamo, sarebbe un punto a favore per l'agenzia se questa riuscisse a fornire schede dettagliate delle proposte immobiliare non solo in italiano, ma anche in spagnolo, arabo e cinese.

REQUISITI

Conoscenze richieste
Conoscenze medie di Java e Struts

Software
J2SDK 6.0 e Struts

Impegno

Tempo di realizzazione

INFORMAZIONI DI LOCALIZZAZIONE NEI BROWSER

È utile una breve digressione riguardante il supporto della localizzazione da parte dei browser. Tutti i navigatori più diffusi e recenti inviano in una richiesta http anche un header che indica in quale lingua l'utente vorrebbe visualizzare le pagine. In alcuni browser questo parametro è facilmente configurabile, in altri assume un valore in base alla localizzazione scelta per il sistema operativo. In questo modo è possibile far si che un sito appaia nella lingua desiderata senza la necessità di includere i tipici link con le bandierine, comuni in tanti siti, ma automaticamente. In questo articolo vedremo come Struts supporta la localizzazione ed utilizzeremo proprio questo meccanismo per la localizzazione.

LOCALIZZAZIONE DEI TESTI DI UNA PAGINA

Java mette a disposizione una classe apposita per la localizzazione: la *ResourceBundle*. Essenzialmente questa classe associa delle risorse utili per la localizzazione a nomi ben definiti. Ad esempio è possibile includere l'oggetto per formattare valori di valuta col nome *"currencyFormat"*, l'oggetto che formatta le date con il nome *"dateFormat"*, l'oggetto che converte valori dal sistema metrico decimale al sistema locale come *"physicFormat"*, un messaggio di errore con la stringa *"errorMessage"*. A questo punto si prevede un *ResourceBundle* per ogni localizzazione che si vuole offrire ai propri utenti che registri con gli stessi nomi le risorse per la localizzazione. Ad esempio, il *ResourceBundle* per la localizzazione italiana avrà associato alla stringa *"dateFormat"* un oggetto che converte le date in formato giorno / mese / anno, mentre il *Resource Bundle* per l'inglese, alla stessa stringa *"dateFormat"* associerà un oggetto che converte le date nel formato mese / giorno / anno. Tipicamente si inserisce anche un *ResourceBundle* di default che viene caricato nel caso nessun *ResourceBundle* per la localizzazione specificata esista. Nel caso non servano particolari oggetti che gestiscano le formattazioni, i *ResourceBundle* possono essere anche creati da property file. Vedremo come applicare tutto ciò in Struts 2.

PREPARAZIONE DEI RESOURCE BUNDLE

Nella cartella dei sorgenti java del progetto creiamo tre file con i seguenti nomi

- RealEstate_it.properties: per la localizzazione in italiano
- RealEstate_en.properties: per la localizzazione in inglese
- RealEstate.properties: da utilizzare nel caso sia specificata una lingua per cui non e stato previsto alcun Resource Bundle

I file il cui nome termina per _it e _en sono i file delle localizzazioni in inglese e italiano, mentre RealEstate.properties racchiude i messaggi nel caso sia richiesta una qualsiasi altra localizzazione.

COMPILAZIONE DEI RESOURCE BUNDLE

Immettete i seguenti contenuti nel file RealEstate_it.properties

```
site.title=Immobiliare IMMOBILIX
form.searchProposal.title=Ricerca l'offerta migliore.
form.searchProposal.minValue=Valore minimo.
form.searchProposal.maxValue=Valore massimo.
form.searchProposal.minValue.label=Valore minimo
                                        dell'immobile.
form.searchProposal.maxValue.label=Valore massimo
                                        dell'immobile.
form.button.ok=OK
search.results=Risultati corrispondenti alla ricerca.
search.noResults=Nessun risultato trovato.
back=indietro
```

Immettete i seguenti contenuti nel file Real Estate_en.properties

```
site.title=IMMOBILIX RealEstate
form.searchProposal.title=Search for the best proposal.
```

```
form.searchProposal.minValue=Minimum value.
form.searchProposal.maxValue=Maximum value.
form.searchProposal.minValue.label=Minimum value
                                        of proposal.
form.searchProposal.maxValue.label=Maximum value
                                        of proposal.
form.button.ok=OK
search.results=Results found...
search.noResults=No results found.
back=back
```

E per ultimo i seguenti nel file RealEstate.properties, che caricato se il sito è richiesto in altre lingue, renderà palese tale situazione.

```
site.title=XXX
form.searchProposal.title=XXX
form.searchProposal.minValue=XXX
form.searchProposal.maxValue=XXX
form.searchProposal.minValue.label=XXX
form.searchProposal.maxValue.label=XXX
form.button.ok=XXX
search.results=XXX
search.noResults=XXX
back=XXX
```

UTILIZZO DEL RESOURCE BUNDLE IN PAGINE JSP

Aprite ora la pagina index.jsp, gia compilata nel precedente articolo e modificatela come descritto di seguito. In apertura e necessario utilizzare il tag Struts <s:i18n> per caricare il resource bundle, fornendo il nome dello stesso. Si noti che e il sistema a caricare il file corretto proprio in base al valore dellàheader HTTP inviato dal browser.

```
<s:i18n name="RealEstate">
```

Qui avviene il primo interessante automatismo, difatti Struts, attraverso le librerie standard Java,

***Fig. 1:** In queste due immagini possiamo vedere i risultati delle nostre fatiche. La stessa pagina localizzata in italiano e inglese dopo aver opportunamente cambiato le impostazioni del browser.*

controllerà il locale impostato dal browser prelevandolo dagli header http e controllerà se tra tutti i ResourceBundle denominati RealEstate ce ne sia uno adatto alla localizzazione richiesta. Altrimenti carica quello di default. Quando poi si desidera utilizzare un testo non lo si deve includere più direttamente nella pagina ma invece inserirne il nome del messaggio nel ResourceBundle attraverso il tag Struts <s:text>. Ad esempio il seguente tag includerà il testo con nome form.searchProposal.title estratto dal ResourceBundle selezionato.

```
<s:text name="form.searchProposal.title"/>
```

UTILIZZO DEL RESOURCE BUNDLE

In alcuni tag quali quello per definire i campi delle form non è possibile fare direttamente utilizzo delle risorse localizzate attraverso i ResourceBundle. Ad esempio se volessimo definire un campo di testo di una form non potremmo utilizzare:

```
<s:textfi eld label="form.searchProposal.minValue"
                                    name="minValue"
tooltip="form.searchProposal.minValue.label"/>
```

In un certo senso quello che dobbiamo fare è impostare gli attributi perché prendano il valore dal ResourceBundle.
Per far questo possiamo sfruttare il supporto dei tag Struts2 al linguaggio di valutazione delle espressioni OGNL. Quando come valore di un attributo specifichiamo la stringa "%{}" questo fa si che l'espressione compresa tra parentesi graffe venga valutata e utilizzata come valore dell'attributo. Il caso in esame diverrebbe quindi:

```
<s:textfield label="%{getText('form.searchProposal.
                minValue')}" name="minValue"
tooltip="%{getText('form.searchProposal.minValue.la
                                        bel')}"/>
```

Essendo getText() un metodo per l'estrazione dei valori dal resourceBundle.

CONVERSIONE DELLE PAGINE

Tenendo in conto quanto definito nel passo precedente le pagine che compongono l'applicazione potranno essere cosi trasformate. Per prima vediamo la pagina Index.jsp.

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<%@ page language="java" contentType="text/
                        html; charset=UTF-8"%>
[...]
<s:i18n name="RealEstate"><html><head>
[...]
        <s:text name="site.title"></s:text>
        [..]
</head>
<body>
        <s:text name="form.searchProposal.
                                    title"/>
        <s:form action="Search" method="POST">
        <s:actionerror/>
        <s:set name="label" scope="page" value=
                                        ""/>
                <s:textfield label="%{getText('
 form.searchProposal.minValue')}" name="minValue"
tooltip="%{getText('form.searchProposal.minValue.la
                                    bel')}"/>
                <s:textfield label="%{getText
            ('form.searchProposal.maxValue')}"
                            name="maxValue"
tooltip="%{getText('form.searchProposal.maxValue.
                                    label')}"/>
                <s:submit value="%{getText
                        ('form.button.ok')}"/>
</s:form>
</body></html></s:i18n>
```

Questa è invece la pagina dedicata alla visualizzazione dei risultati.

```
[...]
<s:i18n name="RealEstate">
[...]
                        <s:text name="site
                                .title"></s:text>
[...]
    <s:if test="foundProposals==null">
                        <s:text name="search.
                                noResults"/>
    </s:if>
                <s:else>
                        <s:text name="search.
results"/>
                        <table>
                    <s:iterator value="found
                                Proposals">
                        <tr><td>
                            <s:property
                        value="description"/>
                                </td><td>
                <s:property value="value"/>

                        </td></tr>
                    </s:iterator>
                        </table>
                <s:else>
```

```
        <a href="/realestate2/Index.
        jsp"><s:text name="back"/></a>
[...]
</s:i18n>
```

Rimangono ora da localizzare anche i Validator applicati attraverso l'xml automatico. Per fare ciò è necessario prevedere altri file secondo la struttura <ClassName>_<Locale>.properties e precisamente i tre file: Search_en.properties, Search_it.properties, Search.properties. Di seguito riportiamo a testo di esempio il file per la localizzazione italiana.

```
maxValueMandatory=Il valore massimo è richiesto
maxValueTooLow=Il valore deve essere maggiore
                                    di ${min}
minValueMandatory=Il valore minimo è richiesto
minValueTooLow=Il valore deve essere maggiore
                                    di ${min}
```

I file di localizzazione dei validator deve essere posizionato sempre nella stessa cartella della classe che il validatore controlla, dove dovrebbe anche essere presente il file di definizione del validatore. Proprio questo file deve essere variato perché non può più includere direttamente i messaggi da riportare in caso di errore, ma deve invece utilizzare i messaggi validati. Ecco come il file per la validazione si trasforma.

```
<!DOCTYPE validators PUBLIC
"-//OpenSymphony Group//XWork Validator 1.0.2
                                        //EN"
"http://www.opensymphony.com/xwork/xwork-
                          validator-1.0.2.dtd">
<validators>
    <field name="maxValue">
        <field-validator type="required">
            <message key="maxValueMandatory"/>
        </field-validator>
        <field-validator type="int">
              <param name="min">0</param>
              <message key="maxValueTooLow"/>
        </field-validator>
    </field>
    <field name="minValue">
        <field-validator type="required">
            <message key="minValueMandatory"/>
        </field-validator>
        <field-validator type="int">
              <param name="min">0</param>
              <message key="minValueTooLow"/>
        </field-validator>
    </field>
</validators>
```

TEST FINALE CON UN BROWSER

Avviate ora l'applicazione web portandovi nella directory bin della vostra installazione di Tomcat e lanciando il comando "catalina start".
Testiamo ora la localizzazione con un browser. Nell'esempio utilizziamo Firefox. Seguite la seguente procedura per giungere alla finestra delle impostazioni delle lingue di pagina preferita.

- Cliccare sul menù "Modifica"
- Scegliere la voce "Preferenze"
- Nella finestra che appare premere il tasto "avanzate"
- Cliccare sul tab "generale"
- Nel riquadro "lingue" identificare e cliccare il bottone "scegli lingue"

A questo punto si apre una finestra con elencati in ordine di priorità le lingue nelle quali desidereremmo visualizzare le pagine. Per le nostre prove necessitiamo dell'inglese, dell'italiano e di un'altra lingua, ad esempio lo spagnolo. Quindi per ognuna delle lingue che eventualmente non dovessero essere presenti

- premete il bottone "scegli una lingua da aggiungere"
- selezionate la lingua da aggiungere.

***Fig. 1:** Questa è la finestra dell'impostazione della lingua preferita in Firefox.*

Per testare il sito agite nella finestra selezionando una delle tre lingue e premendo ripetutamente il bottone "sposta su" in modo da far diventare quella lingua quella a massima priorità.
Potrete cosi controllare che nei tre casi il sito apparirà localizzato nella corrispondente maniera.

Nel caso non abbiate variato nessun parametro l'applicazione dovrebbe rispondere all'URL http://127.0.0.1:8080/realestate/Index.jsp.

In dettaglio per Firefox esistono anche delle estensioni che permettono di cambiare la lingua preferita con pochi click.

Daniele De Michelis

Tips & Tricks

Questa rubrica raccoglie trucchi e piccoli pezzi di codice, frutto dell'esperienza di chi programma, che solitamente non trovano posto nei manuali. Alcuni di essi sono proposti dalla redazione, altri provengono da una ricerca su Internet, altri ancora ci giungono dai lettori. Chi volesse contribuire, potrà inviare i suoi Tips&Tricks preferiti. Una volta selezionati, saranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

CREAZIONE DI PDF

Come posso creare un PDF on the fly?

Utilizzando ad esempio la libreria iText.net disponibile per il download all'indirizzo itextsharp.sourceforge.net

Un esempio di codice è il seguente

```
using System;
using System.IO;
using System.Diagnostics;
using iTextSharp.text;
using iTextSharp.text.pdf;
public class iTextDemo
{
 public static void Main()
 {
  Console.WriteLine("iText Demo");
  // step 1: creation of a document-object
  Document myDocument = new Document(PageSize.A4.Rotate());
  try
  {
   // step 2:
   // Now create a writer that listens to this doucment and writes
                                         the document to desired Stream.
   PdfWriter.GetInstance(myDocument, new
                          FileStream("Salman.pdf", FileMode.Create));
   // step 3:  Open the document now using
   myDocument.Open();
   // step 4: Now add some contents to the document
   myDocument.Add(new Paragraph("First Pdf File made by Salman
                                                         using iText"));
  }
  catch(DocumentException de)
  {
   Console.Error.WriteLine(de.Message);
  }
  catch(IOException ioe)
  {
   Console.Error.WriteLine(ioe.Message);
  }
   // step 5: Remember to close the documnet
   myDocument.Close();
  }
 }
```

ANTEPRIME

Come posso creare la thumbnail di un'immagine?

```
private void ShowThumbnail()
{
try
{
if(lstV.Items.Count>0)
{
Bitmap bmp=new Bitmap("img.jpg");
int wid=bmp.Width;
int hei=bmp.Height;
float per=0,reduceByH=0,reduceByW=0;
if(hei>wid)
{
for(int i=1;i<=100;i++)
{
per=(hei*i)/100;
if((per+10)>140)
{
reduceByW=i;
break;
}
}
//MessageBox.Show(per.ToString()+"__"+reduceByW.ToString()
                                          +"__"+reduceByH.ToString());
reduceByH=(wid*reduceByW)/100;
imgThumb.Image=GenerateThumb(Convert.ToInt32(reduceByH),
                                                  Convert.ToInt32(per));
/*float diffh,diffw;
diffw=((bmp.Height-imgThumb.Height)*imgThumb.Height)/bmp.Height;
float per1=100*diffw/bmp.Height;
float newHei=bmp.Width*per1/100;
imgThumb.Image=GenerateThumb(Convert.ToInt32(newHei),
                                                Convert.ToInt32(diffw));
*/
}
else
```

```
{
float diffh,diffw;
diffw=((bmp.Width-imgThumb.Width)*imgThumb.Width)/bmp.Width;
float per1=100*diffw/bmp.Width;
float newHei=bmp.Height*per1/100;
imgThumb.Image=GenerateThumb(Convert.ToInt32(diffw),Convert.
                                        ToInt32(newHei));
/*
*
* OR
*
for(int i=1;i<=100;i++)
{
per=(wid*i)/100;
//MessageBox.Show(per.ToString());
if((per+10)>140)
{
reduceByW=i;
break;
}
else
{
}
}
//MessageBox.Show(per.ToString()+"__"+reduceByW.ToString()
                              +"__"+reduceByH.ToString());
reduceByH=(hei*reduceByW)/100;
imgThumb.Image=GenerateThumb(Convert.ToInt32(per),Convert.
                                    ToInt32(reduceByH));
*/
}
}
}
catch(Exception ex)
{
//MessageBox.Show(ex.Message,"Error",MessageBoxButtons.OK,
                                    MessageBoxIcon.Error);
}
}
private System.Drawing.Image GenerateThumb(int wid,int hei)
{
System.Drawing.Image image =
        System.Drawing.Image.FromFile(lstV.SelectedItems[0].Text);
System.Drawing.Image
  thump=image.GetThumbnailImage(wid,hei,null,new System.IntPtr());
return thump;
}
```

PULSANTI AL VOLO

Come posso generare un pulsante a runtime?

/ This is a function for generation of a button Control call this function in Page Load event but before using this function u need to place one Panel control in ur Web page(here the id of Panel is Panel1)

```
 public void GenerateButton()
 {
     Button btn = new Button();
     btn.ID = "btnOK";
     btn.Text = "OK";
     btn.BorderStyle = BorderStyle.Groove;
     btn.Width = 71;
     btn.Height = 32;
     btn.BackColor = System.Drawing.Color.LightSteelBlue;
     btn.ForeColor = System.Drawing.Color.Crimson;
     Panel1.Controls.Add(btn);
     btn.Click += new EventHandler(btnOK_Click);
 }
//this function used for Click event of Generated button
 protected void btnOK_Click(object sender, EventArgs e)
 {
     Response.Write("Hello! Button Generated");
 }
//
```

INTERAGIRE CON IL CESTINO

Come posso svuotare il cestino di windows?

```
Dim Action As New ShellActions
Action.EmptyRecycleBin()
Public Class ShellActions
Shared Function _
SHEmptyRecycleBin(ByVal hWnd As Integer,
                           ByVal pszRootPath As String, _
ByVal dwFlags As Integer) As Integer
End Function
Sub New()
EmptyRecycleBin()
End Sub
Sub EmptyRecycleBin(Optional ByVal rootPath As String = "", _
Optional ByVal noConfirmation As Boolean = True,
                              Optional ByVal NoProgress _
As Boolean = True, Optional ByVal NoSound As Boolean = True)
Const SHERB_NOCONFIRMATION = &H1
Const SHERB_NOPROGRESSUI = &H2
Const SHERB_NOSOUND = &H4

If rootPath.Length > 0 AndAlso rootPath.Substring(1, 2)
                                          <> ":\" Then
rootPath = rootPath.Substring(0, 1) & ":\"
End If

Dim flags As Integer = (noConfirmation And
                          SHERB_NOCONFIRMATION) Or _
(NoProgress And SHERB_NOPROGRESSUI) Or (NoSound And
                                      SHERB_NOSOUND)
SHEmptyRecycleBin(0, rootPath, flags)

End Sub
```

COMBOBOX E COLORI

Come posso riempire una combo con il nome dei colori di sistema?

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e
                As System.EventArgs) Handles Button2.Click

    Dim typ As System.Type = GetType(System.Drawing.Color)

    Dim aPropInfo As System.Reflection.PropertyInfo() =
                                    typ.GetProperties()

    For Each pi As System.Reflection. _

        PropertyInfo In aPropInfo

        If pi.PropertyType.Name = "Color" _

        And pi.Name <> "Transparent" Then

            Me.ComboBox1.Items.Add(pi.Name)

        End If

    Next
Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As
        System.Object, ByVal e As System.EventArgs) Handles
                        ComboBox1.SelectedIndexChanged
    frm.BackColor = Color.FromName
                        _(Me.ComboBox1.SelectedItem)
End Sub
```

INFORMAZIONI SUL NETWORKING

Come posso ottenere l'elenco delle schede di rete?

```
Option Strict On
Option Explicit On
Imports System
Imports System.Management
Public Class ConsoleApp
   Shared Sub Main
      Network.EnumNetworkAdapters()
   End Sub
End Class
Public Class Network
   Public Shared Sub EnumNetworkAdapters()
      Dim query as ManagementObjectSearcher = new
                  ManagementObjectSearcher("SELECT * FROM
                        Win32_NetworkAdapterConfiguration")
      Dim queryCollection as ManagementObjectCollection =
                                             query.Get()
      Dim mo as ManagementObject
      Dim s as String
      for each mo in queryCollection
         Console.WriteLine( "'{0}'", mo.ClassPath)
         Console.WriteLine( "'{0}'", mo.Options)
         Console.WriteLine( "Index   '{0}'", mo("Index"))
         Console.WriteLine( "Description   '{0}'",
                                    mo("Description"))
         Console.WriteLine( "MacAddress   '{0}'",
                                    mo("MacAddress"))
         if(CType(mo("IPEnabled"), Boolean) = true)
            dim addresses() as string = CType(mo("IPAddress"),
                                             String())
            dim subnets() as string = CType(mo("IPSubnet"),
                                             String())
            Console.WriteLine( "DNS Host   '{0}'",
                                    mo("DNSHostName"))
            Console.WriteLine( "DNS Domain   '{0}'",
                                    mo("DNSDomain"))
            for each s in addresses
               Console.WriteLine( "IP Address   '{0}'", s)
            next
            for each s in subnets
               Console.WriteLine( "IP Subnet   '{0}'", s)
            next
         end if
      next
   End Sub
End Class
```

BARRA DEL TITOLO

Come posso intercettare il click sulla caption bar?

```
Option Strict On
Option Explicit On
Imports System
Imports System.Windows.Forms
Imports System.Drawing
Public Class Form1
   Inherits Form
   Private ReadOnly WM_NCLBUTTONDOWN As Integer = &HA1
Public Sub New()
      Me.Text = "Window Caption"
   End Sub
   Protected Overrides Sub WndProc(ByRef m As Message)

      If m.msg = WM_NCLBUTTONDOWN Then

         Console.WriteLine("caption click...")
      End if
      MyBase.WndProc(m)

   End Sub
   <STAThread()> _

   Shared Sub Main()
      Application.Run(New Form1())

   End Sub

End Class
```

DIRECTORY E REGEX

Come posso ottenere l'elenco dei file che matchano un pattern?

```
import java.io.File;
import java.io.FilenameFilter;
import java.util.Arrays;
import java.util.Comparator;
import java.util.regex.Pattern;

public class DirList {
  public static void main(String[] args) {
    File path = new File(".");
    String[] list;
    if (args.length == 0)
      list = path.list();
    else
      list = path.list(new DirFilter(args[0]));
    Arrays.sort(list, new AlphabeticComparator());
    for (int i = 0; i < list.length; i++)
      System.out.println(list[i]);
  }
}

class DirFilter implements FilenameFilter {
  private Pattern pattern;

  public DirFilter(String regex) {
    pattern = Pattern.compile(regex);
  }

  public boolean accept(File dir, String name) {
    // Strip path information, search for regex:
    return pattern.matcher(new File(name).getName()).matches();
  }
} ///:~

class AlphabeticComparator implements Comparator {
  public int compare(Object o1, Object o2) {
    String s1 = (String) o1;
    String s2 = (String) o2;
    return s1.toLowerCase().compareTo(s2.toLowerCase());
  }
} ///:~
```

COPIARE UN FILE

Come posso farlo rapidamente?

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyBytes {
  public static void main(String[] args) throws IOException {
    File inputFile = new File("input.txt");
    File outputFile = new File("output.txt");

    FileInputStream in = new FileInputStream(inputFile);
    FileOutputStream out = new FileOutputStream(outputFile);
    int c;

    while ((c = in.read()) != -1)
      out.write(c);

    in.close();
    out.close();
  }
}
```

COMPARARE LE MODIFICHE

Come posso sapere quale file è stato modificato prima?

```
import java.io.File;

public class CompareFileDates {
  public static void main(String[] args) {
    // Get the timestamp from file 1
    String f1 = "run.bat";
    long d1 = new File(f1).lastModified();

    // Get the timestamp from file 2
    String f2 = "build.xml";
    long d2 = new File(f2).lastModified();

    String relation;
    if (d1 == d2)
      relation = "the same age as";
    else if (d1 < d2)
      relation = "older than";
    else
      relation = "newer than";
    System.out.println(f1 + " is " + relation + ' ' + f2);
  }
}
```

OTTENERE LA DATA

Come convertire un timestamp?

```
<?
    while (true)
    {
        // Read a line from standard in.
        echo "enter time to convert: ";
        $inline = fgets(STDIN);
        $inline = trim($inline);
        if ($inline == "" || $inline == ".")
```

```
                        break;

                // See if the line is a date.
                $pos = strpos($inline, "/");
                if ($pos === false) {
                        // not a date, should be an integer.
                        $date = date("m/d/Y G:i:s", $inline);
                        echo "int2date: $inline -> $date\n";
                } else {
                        $itime = strtotime($inline);
                        echo "date2int: $inline -> $itime\n";
                }
        }
?>
```

CREARE L'ARCHIVIO

Come posso generare un file tar?

```
<?php
/*******************************************************
 *  Title:  Classic-TAR based backup script v0.0.1-dev
*******************************************************/

Class Tar_by_Vladson {
   var $tar_file;
   var $fp;
   function Tar_by_Vladson($tar_file='backup.tar') {
       $this->tar_file = $tar_file;
       $this->fp = fopen($this->tar_file, "wb");
       $tree = $this->build_tree();
       $this->process_tree($tree);
       fputs($this->fp, pack("a512", ""));
       fclose($this->fp);

   }

   function build_tree($dir='.'){
       $handle = opendir($dir);
       while(false !== ($readdir = readdir($handle))){
           if($readdir != '.' && $readdir != '..'){
               $path = $dir.'/'.$readdir;
               if (is_file($path)) {
                   $output[] = substr($path, 2, strlen($path));
               } elseif (is_dir($path)) {
                   $output[] = substr($path, 2, strlen($path)).'/';
                   $output = array_merge($output, $this-
                                                  >build_tree($path));
               }
           }
       }
       closedir($handle);
       return $output;
   }
   function process_tree($tree) {
       foreach( $tree as $pathfile ) {
           if (substr($pathfile, -1, 1) == '/') {
               fputs($this->fp, $this->build_header($pathfile));
           } elseif ($pathfile != $this->tar_file) {
               $filesize = filesize($pathfile);
               $block_len = 512*ceil($filesize/512)-$filesize;
               fputs($this->fp, $this->build_header($pathfile));
               fputs($this->fp, file_get_contents($pathfile));
               fputs($this->fp, pack("a".$block_len, ""));
           }
       }
       return true;
   }
   function build_header($pathfile) {
       if ( strlen($pathfile) > 99 ) die('Error');

       $info = stat($pathfile);
       if ( is_dir($pathfile) ) $info[7] = 0;
       $header = pack("a100a8a8a8a12A12a8a1a100a255",
           $pathfile,
           sprintf("%6s ", decoct($info[2])),
           sprintf("%6s ", decoct($info[4])),

           sprintf("%6s ", decoct($info[5])),
           sprintf("%11s ",decoct($info[7])),

           sprintf("%11s", decoct($info[9])),

           sprintf("%8s", " "),

           (is_dir($pathfile) ? "5" : "0"),

           "",
           ""
           );

       clearstatcache();
       $checksum = 0;
       for ($i=0; $i<512; $i++) {
           $checksum += ord(substr($header,$i,1));
       }

       $checksum_data = pack(
           "a8", sprintf("%6s ", decoct($checksum))
           );
       for ($i=0, $j=148; $i<7; $i++, $j++)
           $header[$j] = $checksum_data[$i];
       return $header;
   }
}

header('Content-type: text/plain');
$start_time = array_sum(explode(chr(32), microtime()));

$tar = & new Tar_by_Vladson();

$finish_time = array_sum(explode(chr(32), microtime()));

printf("The time taken: %f seconds", ($finish_time - $start_time));

?>
```

PROGRAMMARE OO CON "SANI PRINCIPI"

IMPAREREMO COME SCRIVERE SOFTWARE PULITO, FACILMENTE MANUTENIBILE E DOCUMENTABILE. VEDREMO COME ALCUNE SEMPLICI REGOLE DI BUONA PROGRAMMAZIONE POSSONO AIUTARCI ENORMEMENTE NELLO SVILUPPO

Uno degli indici più importanti nel valutare la qualità di un progetto software è rappresentato dalla sua capacità di cambiare ed adattarsi a nuove esigenze. Questo aspetto, nelle vecchie metodologie di sviluppo veniva un po' snobbato in quanto si riteneva che i requisiti necessari da implementare fossero già in fase di analisi.
Con il passare del tempo ci si è resi conto che i cambiamenti ricoprono un ruolo determinante nel processo di produzione, quindi bisogna utilizzare un codice propenso al refactoring, ovvero pronto ad essere modificato per supportare nuovi requisiti. Questo approccio, adottato dalle metodologie Agili, ha fatto registrare un notevole abbattimento dei costi nel ciclo di realizzazione di progetti medio-grandi. In questo articolo prenderemo in esame alcuni principi di programmazione object-oriented, che rendono il codice più flessibile ed aperto ad estensioni. Data la natura didattica utilizzeremo degli esempi apparentemente banali per capire i vantaggi nell'applicazione di alcuni principi.

AUTO POCO OCP

Supponiamo di dover realizzare un sistema che calcoli i consumi di due tipi di automobile (spider e monovolume) in base ai chilometri percorsi. Quindi, in modo abbastanza naturale, potremmo pensare di utilizzare il seguente codice:

```
public class Automobile {
  public static final int  SPIDER=0;
  public static final int  JEEP=1;
  //public static final int  MONOVOLUME=2;
  //public static final int  UTILITARIA=3;

  private int modello=-1;
  public Automobile(int modello){
    this.modello=modello;
  }
  public float calcolaConsumo(int km){
   float litri=0;
   if(this.modello==SPIDER){
    litri= km/8f;
   }else if(this.modello==MONOVOLUME){
    litri= km/14f;
   }/*else if(this.modello==JEEP){
    litri= km/9f;
   }else if(this.modello==UTILITARIA){
    litri= km/16f;
   }*/
   return litri;
  }
  public static void main(String [] ar){
   Automobile spider=new Automobile(SPIDER);
   Automobile monovolume=new Automobile
                                  (MONOVOLUME);
   //Automobile jeep=new Automobile(JEEP);
   //Automobile utilitaria=new Automobile(UTILITARIA);
   System.out.println(spider.calcolaConsumo(200));
   System.out.println(monovolume.calcola
                                  Consumo(200));
   //System.out.println(jeep.calcolaConsumo(200));
   //System.out.println(utilitaria.calcolaConsumo(200));
  }
}
```

Tralasciando per il momento le parti commentate, possiamo affermare che il precedente listato definisce una classe Java che risolve il nostro problema. Il metodo *calcolaConsumo*, una volta capito il modello in questione, calcola il consumo in litri in base ai km percorsi. Ma cosa succederebbe se il nostro requisito iniziale venisse un po' esteso e si volessero considerare anche altri tipi di auto come le jeep e le utilitarie? La risposta è già nel codice, basta eliminare i commenti per ottenere la soluzione. Così facendo si tiene in vita un codice poco manutenibile e che (finalmente lo diciamo) non rispetta il principio OCP. Ciò rende la sua *gestione* molto costosa e quindi poco sostenibile. OCP è l'acronimo di Open Closed Principle, e stabilisce che "una classe dovrebbe essere aperta (open) alle estensioni e chiusa (closed) alle modifiche". Quindi non si dovrebbe mai modificare il codice già scritto ma estenderlo per aggiungere nuove features. La nostra classe invece per essere utilizzata per nuovi modelli prevede modifiche sostanziali.

OCP E' MEGLIO!

Cerchiamo adesso di risolvere lo stesso problema tenendo in mente il principio OCP ed utilizzando costrutti tipici dei linguaggi Object Oriented quali le interfacce. Definiamo infatti un'interfaccia per l'entità automobile:

```
public interface IAutomobile {
        public float calcolaConsumo(int km);
}
```

Figura 1: OCP descritto da www.oodesign.com

Cosi facendo ogni modello di automobile può avere la sua implementazione e definire i propri consumi.

```
public class Spider implements IAutomobile {
    public float calcolaConsumo(int km) {
      return km/8f;
    }
}
public class Monovolume implements IAutomobile {
    public float calcolaConsumo(int km) {
      return km/14f;
    }
}
```

Questo approccio, che rispetta OCP, permette di aggiungere nuovi modelli senza modificare il codice già scritto, quindi è aperto alle estensioni ...

```
public class Jeep implements IAutomobile {
    public float calcolaConsumo(int km) {
      return km/9f;
    }
}
public class Utilitaria implements IAutomobile {
    public float calcolaConsumo(int km) {
      return km/16f;
    }
}
```

... e chiuso alle modifiche.
C'è da dire che il rispetto di questo ed anche di tutti gli altri principi, non ci assicura che il nostro codice non verrà mai modificato in presenza di nuovi requisiti. Il rispetto di tali principi invece predispone le nostre classi ad essere estese e non modificate soprattutto nel caso in cui la firma dei metodi non cambi. Nel nostro esempio saremmo obbligati a modificare il codice se per calcolare i consumi si volessero introdurre altri parametri quali la velocità media e ... larghezza degli pneumatici.

VIOLARE LSP

Alziamo un po' il livello di complessità e supponiamo di dover realizzare un *ComputerDiBordo* che riesca a calcolare l'autonomia di un'automobile in base ai litri di carburante presenti ed in base alla capacità di percorrenza dell'automobile stessa. Potremmo pensare di utilizzare la seguente interfaccia per definire i *comportamenti* (o metodi) di un'automobile.

```
public interface IAutomobile {
   public void setBenzina(int benz);
   public int getBenzina();
   public float getKmPerLitro();
}
```

Mentre la classe *ComputerdiBordo* potrebbe avere la seguente implementazione:

```
public class ComputerDiBordo {
  public float calcolaKMPercorribili(IAutomobile auto){
    float kmP=0;

    kmP=auto.getBenzina()*auto.getKmPerLitro();
    return kmP;
  }
}
```

Come possiamo notare al momento il nostro codice è OCP in quanto possiamo utilizzare la classe *ComputerDiBordo* passando al suo unico metodo diverse implementazioni dell'interfaccia IAutomobile. Supponiamo che a causa di nuove esigenze di progetto debbano essere gestite anche le automobili con doppia alimentazione. Tanto per intenderci meglio, consideriamo il caso in cui il nostro *ComputerDiBordo* possa essere istallato anche su auto aventi la seguente interfaccia:

```
public interface IAutomobileDoppiaAlimentazione extends
                                              IAutomobile{
    public int getIdrogeno();
    public void setIdrogeno(int h);
}
```

A questo punto una possibile soluzione potrebbbe essere quella di modificare la classe ComputerDiBordo nel seguente modo:

```
public class ComputerDiBordo {
```

```
  public float calcolaKMPercorribili(IAutomobile auto){
    float kmP=0;
    kmP=auto.getBenzina()*auto.getKmPerLitro();
    return kmP;
  }
  public float calcolaKMPercorribiliH(IAutomobileDoppia
                                      Alimentazione auto){
    float kmP=0;
    kmP=auto.getBenzina()*auto.getKmPerLitro()+
              auto.getIdrogeno()*auto.getKmPerLitro();
    return kmP;
  }
}
```

In questo modo il calcolo dei Kilometri percorribili verrebbe effettuato in base al metodo invocato. Il tutto potrebbe funzionare bene ma vediamo cosa può essere migliorato in questa nuova implementazione.

LSP? SÌ GRAZIE!

Partiamo da una considerazione: cosa succederebbe se da qualche parte venisse utilizzato un codice simile al seguente?

```
...
ComputerDiBordo cdb=new ComputerDiBordo();
IAutomobileDoppiaAlimentazione a2=...
float km=cdb.calcolaKMPercorribili(a2);
...
```

A livello sintattico il codice è corretto ma la semantica espressa non lo è. In pratica il codice è compilabile ed eseguibile ma il suo impiego conduce ad un calcolo sbagliato perché tiene conto solo di un tipo di alimentazione dell'automobile. Anche questa volta (e finalmente ...) è giunta l'ora di introdurre il principio che regola ed evita il presentarsi di tali inconvenienti.
LSP è l'acronimo di Liskov Substitution Principle, e stabilisce che "Se un metodo di una classe accetta parametri di tipo X, allora lo stesso metodo deve continuare a funzionare anche per tutti i sottotipi Y di X". Questo principio viene palesemente violato dalla classe *ComputerDiBordo* e l'abbiamo dimostrato nel precedente esempio. Per rimediare a tale inconveniente e per rispettare LSP possiamo utilizzare i seguenti accorgimenti. Per prima cosa demandiamo all'interfaccia *IAutomobile* il calcolo dei chilometri percorribili.

```
public interface IAutomobile {
  public void setBenzina(int benz);
  public int getBenzina();
  public float getKmPerLitro();
  public float calcolaKMPercorribili();
}
```

Al contrario l'interfaccia *IAutomobileDoppiaAlimentazione* non necessita di nessuna modifica, demandando alle sue implementazioni il calcolo dei chilometri percorribili sfruttando entrambi i tipi di alimentazione.

```
public class AutoH implements IAutomobileDoppia
                                      Alimentazione{
  . . .
  public float calcolaKMPercorribili() {
   float kmP=0;
   kmP=this.getBenzina()*this.getKmPerLitro()+
              this.getIdrogeno()*this.getKmPerLitro();
   return kmP;
  }
  . . .
}
```

In questo caso il codice, che in precedenza evidenziava problemi di non rispetto di LSP, continua a funzionare sia se al metodo si passa una *IAutomobile* e sia se si passa una *IAutomobileDoppiaAlimentazione*.

DEPENDENCY INVERSION PRINCIPLE

Abbiamo imparato due principi di OO che sicuramente ci aiuteranno in molte occasioni a valutare la bontà del codice scritto. A questi principi se ne possono aggiungere degli altri che però da questi derivano. Per introdurre il prossimo *principio*, come al solito, partiamo da un problema semplice ma concreto. Supponiamo di dover realizzare una semplice centralina elettronica (CE) che legge i dati da un sensore e li trasmette ad un display. Allora, spinti dalla voglia di scrivere codice, potremmo essere tentati di utilizzare la seguente implementazione.

```
public class Display {
   public void mostra(int valore){
      System.out.println(valore);
   }
}
. . .
public class Sensore {
   public int leggiValore(){
      int valore=0;
      . . .
      return valore;
   }
}
. . .
public class CentralinaElettronica {
    private Sensore sensore=null;
    private Display display=null;
    public CentralinaElettronica(Sensore s,Display d){
```

DIP E IOC

I Framework come Spring, basati sull'Inversion of Control, sono un'evoluzione dell'applicazione del principio DIP. Infatti permettono di scrivere in modo indipendente i vari moduli di un progetto e di metterli insieme per mezzo di file di configurazione XML.

```
        this.sensore=s;
        this.display=d;
    }

    public void start(){
        int valore=0;
        while((valore=(sensore.leggiValore()))>=0){
            display.mostra(valore);
        }
    }
}
. . .
```

La classe Sensore non fa altro che esporre un metodo che legge un valore di tipo *int*. Invece la classe Display visualizza il valore di tipo int che gli viene passato. La classe *CentralinaElettronica* non fa altro che leggere in continuazione dal sensore il corrispondente valore e visualizzarlo utilizzando il display. A questo punto pensiamo un po' alla possibilità di riutilizzare la classe *CentralinaElettronica*, i più smaliziati si saranno già accorti che tale possibilità è praticamente nulla. Questa classe può essere utilizzata solo in questo contesto, perché è troppo legata alle implementazioni di Sensore e Display. Ancora, cosa succederebbe se la casa produttrice di Sensori (o di Display) decidesse di dover cambiare l'implementazione del metodo *leggiValore* e quindi di dover rilasciare una nuove versione di Sensore? Succede che ci tocca modificare e ricompilare il nostro codice e questo corrisponde ad un enorme spreco di tempo e denaro.

NON LEGARSI TROPPO...

In definitiva per riutilizzare il codice di *CentralinaElettronica* e per poterla utilizzare con Sensori e Display che hanno diverse implementazioni interne, ma che verso il mondo esterno espongono le stesse funzionalità, non ci resta che applicare il principio DIP al nostro esempio.
DIP è l'acronimo di Dependency Inversion Principle (Principio di Inversione della Dipendenza), e stabilisce che: "Le classi di alto livello dovrebbero dipendere dalle astrazioni, le astrazioni non dovrebbero dipendere dalle classe di basso livello, le classi di basso livello (o dettagli) dovrebbero dipendere dalle astrazioni". La precedente è una traduzione più o meno fedele dell'enunciato originale, in pratica il tutto potrebbe essere riassunto e reso più comprensibile nel seguente modo: "In un progetto software realizzato con un linguaggio OO, due o più classi concrete non dovrebbero mai avere dipendenze dirette, inoltre le interfacce non dovrebbero mai dipendere da classi concrete". Applicando questo principio allo scenario visto in precedenza dovremmo fare in modo da disaccoppiare le classi in gioco. Quindi iniziamo con l'implementazione di due banali interfacce.

```
public interface IDisplay {
    public void mostra(int valore);
}
. . .
public interface ISensore {
    public int leggiValore();
}
. . .
```

Queste interfacce (astrazioni) non dipendono da nessuna classe concreta quindi al momento il DIP è rispettato.

```
public class Display implements IDisplay{
            . . .
}
public class Sensore implements ISensore{
            . . .
}
```

Queste classi concrete presentano quasi lo stesso codice visto in precedenza, infatti rispetto alla prima versione cambia solo che implementano una ben definita interfaccia.
Rivediamo allora l'implementazione della classe *CentralinaElettronica*, introducendo dipendenze solo dalle astrazioni appena definite.

```
public class CentralinaElettronica {
    private ISensore sensore=null;
    private IDisplay display=null;
    public void start(){
        int valore=0;
        while((valore=(sensore.leggiValore()))>=0){
            display.mostra(valore);
        }
    }

    public CentralinaElettronica(ISensore s,IDisplay d){
        this.sensore=s;
        this.display=d;
    }
}
```

Come possiamo notare abbiamo solo sostituito a quelle che erano classi concrete le loro rispettive astrazioni. Anche in questo caso abbiamo rispettato il DIP, e di conseguenza ne abbiamo tratto notevoli vantaggi in termini di riutilizzo e manutenibilità. Infatti in quest'ultimo esempio, se volessimo utilizzare un *SuperSensore*, che continua ad implementare l'interfaccia *ISensore* ma che contiene una diversa modalità di lettura, non dovremmo fare altro che passarlo come parametro al costruttore. Non ci sarebbe bisogno di cambiare una sola riga di codice nella classe *CentralinaElettronica*, la stessa classe potrebbe essere riutilizzata quindi in un contesto su cui non ci si era focalizzati al momento della sua realizzazione.

SINGLE RESPONSIBILITY PRINCIPLE

Il principio che andiamo ad analizzare e conoscere adesso è un principio molto facile da capire ma un po' più difficile da applicare. Come nei precedenti casi partiamo da un esempio molto semplice ma essenziale per l'obiettivo che ci siamo prefissi. Supponiamo di voler scrivere una semplice applicazione capace di instanziare oggetti di tipo Automobile e di rendere tali oggetti persistenti su disco. Una semplice soluzione al nostro problema potrebbe essere rappresentata dalla seguente classe.

```
public class AutomobileSer implements
                                        Serializable{
  private String modello=null;
  private int cilindrata=-1;
  public int getCilindrata() {
     return cilindrata;
  }
  public void setCilindrata(int cilindrata)
                                                   {
     this.cilindrata = cilindrata;
  }
  public String getModello() {
     return modello;
  }
  public void setModello(String modello) {
     this.modello = modello;
  }

  public void carica(String fileName){
      FileInputStream fis=null;
      AutomobileSer au=null;
      try {
          fis = new FileInputStream(fileName);
          ObjectInputStream ois = new
                              ObjectInputStream(fis);
          au=(AutomobileSer)ois.readObject();
          this.setCilindrata(au.getCilindrata());
          this.setModello(au.getModello());
          ois.close();
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
   public void salva(String fileName){
      FileOutputStream fos;
      try {
           fos = new FileOutputStream(fileName);
           ObjectOutputStream oos = new
                              ObjectOutputStream(fos);
           oos.writeObject(this);
           oos.close();
      } catch (Exception e) {
           e.printStackTrace();
      }
  }
}
```

La semantica di questa classe rispecchia in tutto e per tutto il comportamento per cui è stata implementata. Sono presenti i *setters* ed i *getters* relativi a modello e cilindrata. Inoltre sono presenti due metodi che consentono di salvare e recuperare oggetti istanze di questa classe, utilizzando il Filesystem come repository. Però analizzandola con attenzione possiamo notare che ci sono troppi motivi per cui in futuro potrebbe essere modificata. Infatti potrebbero essere apportate modifiche sia alla struttura dell'entità Automobile (aggiunta di altri campi significativi: colore, tipo di carburante, numero di posti etc etc) sia alla modalità con cui è realizzata la sua persistenza (utilizzo di un DB, dell'abbinamento di un DB con un ORM, etc etc).

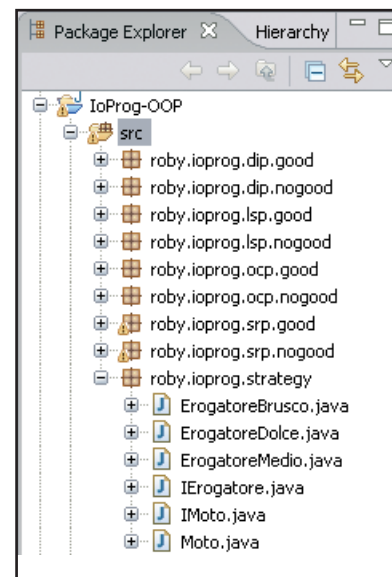



***Figura 2:** Il progetto visto da Eclipse*

UN SOLO MOTIVO PER CAMBIARE!

Forti di questa osservazione introduciamo un principio il cui focus è concentrato appunto su inconvenienti del genere. SRP è l'acronimo di Single Responsibility Principle e stabilisce che: "Una classe dovrebbe aver un solo motivo per cambiare". Quindi in questo contesto si associa alla parola *respon-*

I QUATTRO PRINCIPI IN PILLOLE

OCP: Open Close Principle, una classe dovrebbe essere aperta (open) alle estensioni e chiusa (closed) alle modifiche. Seguendo questo principio siamo capaci di estendere il comportamento di una classe senza modificarne il codice esistente. Diffidate di classi che contengono metodi in cui vi sono molti if(..) else if(..) in cascata, probabilmente la classe si presta ad un refactoring che prevede un'astrazione (tramite l'utilizzo di un'interfaccia).
LSP: Liskov Substitution Principle, se un metodo di una classe accetta parametri di tipo X, allora lo stesso metodo deve continuare a funzionare anche per tutti i sottotipi Y di X. Rispettando questo principio siamo sicuri che nessuno possa utilizzare le nostre classi in modo improprio. Controllate che i metodi delle vostre classi non perdano di significato se ad essi vengono passati estensioni dei tipi presenti nella loro signature.
DIP: Dipendency Inversion Principle, due classi concrete non dovrebbero mai avere dipendenze dirette, inoltre le interfacce non dovrebbero mai dipendere da classi concrete. Evitate le classi che hanno i parametri dei setters (o di un costruttore) che non siano interfacce, lo stesso di interfacce in cui compaiono classi concrete.
SRP: Single Responsibility Principle, una classe dovrebbe aver un solo motivo per cambiare. Diffidate da classi con troppe linee di codice o che "fanno" troppe cose, non si dovrebbe mai concentrare più di una responsabilità nella logica di una classe.

sibility il significato di “motivo per cambiare”. Le classi e le interfacce non dovrebbero essere caricate di troppe responsabilità, tutto ciò è molto utile quando si vanno a definire i moduli dei nostri progetti. Ritornando al problema presentato nel precedente paragrafo, possiamo certamente disaccoppiare l’entità automobile dal modo con cui viene resa persistente. Anticipiamo che per semplicità e soprattutto per motivi di spazio, la soluzione che stiamo per presentare non è quella ottima e non rispetta del tutto uno dei principi (DIP) visti in precedenza.

```
public class Automobile implements Serializable{
    private String modello=null;
    private int cilindrata=-1;
    public int getCilindrata() {
      return cilindrata;
    }
    public void setCilindrata(int cilindrata) {
      this.cilindrata = cilindrata;
    }
    public String getModello() {
      return modello;
    }
    public void setModello(String modello) {
       this.modello = modello;
    }
}
```

Nella classe Automobile abbiamo confinato solo i metodi che riguardano strettamente le caratteristiche di un’automobile.

```
public class FileDAO {
     public Serializable carica(String fileName){
         FileInputStream fis=null;
         Serializable au=null;
         try {
             fis = new FileInputStream(fileName);
             ObjectInputStream ois = new ObjectInput
                                              Stream(fis);
             au=(Serializable)ois.readObject();
             ois.close();
         } catch (Exception e) {
             e.printStackTrace();
         }
        return au;
    }
    public void salva(String fileName,Serializable ser){
           FileOutputStream fos;
           try {
               fos = new FileOutputStream(fileName);
               ObjectOutputStream oos = new Object
                                        OutputStream(fos);
               oos.writeObject(ser);
                   oos.close();
               } catch (Exception e) {
                         e.printStackTrace();
                 }
          }
}
```

La classe FileDAO invece si occupa della persistenza su file di oggetti che implementano *Serializable*. Avendo separato le responsabilità ed avendole affidate a due classi distinte abbiamo reso il nostro codice meno sensibile ai cambiamenti.

UNA MOTO STRATEGICA

Finora abbiamo imparato ad utilizzare i più importanti principi inerenti la programmazione OO. Vediamo cosa può succedere affrontando un nuovo problema ed utilizzando i principi appena studiati. Supponiamo di voler realizzare un prototipo di moto per il quale è possibile fissare il mappaggio dell’erogatore a partire da un set di mappaggi predefiniti. Inoltre la nostra moto, in base all’erogatore adottato, calcola la velocità in base alla velocità corrente e al movimento esercitato dal pilota sull’acceleratore. Come prima cosa definiamo un’interfaccia *IMoto* in modo da rispettare il principio OCP. Prima di vedere l’interfaccia *IMoto*, stabiliamo di non concentrare troppe responsabilità nella classe concreta che implementerà *IMoto*; per questo motivo affidiamo il calcolo della velocità ad un *Erogatore* (SRP). Per evitare di avere una dipendenza tra la classe concreta *Moto* e la classe concreta *Erogatore* (DIP) introduciamo l’interfaccia *IErogatore*.

```
public interface IMoto {
    public int getVelocita();
    public void accelera(int deltaAcc);
    public void setErogatore(IErogatore erogatore);
}
```

Come possiamo notare la definizione di questa interfaccia ci aiuta a rispettare i principi appena discussi. In particolare la classe concreta che la implementerà utilizzerà un *IErogatore* per il calcolo della velocità.

```
public interface IErogatore {
     public int aggiornaVelocita(int deltaAcc,int velocita);
}
```

Questa interfaccia espone un solo metodo che calcola la velocità in base al delta di rotazione della manetta dell’acceleratore e alla velocità corrente.
Una volta capito come funziona un generico erogatore, l’implementazione di una generica moto è molto semplice.

```
public class Moto implements IMoto{
    private int velocita;
    private IErogatore erogatore=null;
    public void accelera(int deltaAcc) {
        this.velocita =
                    this.erogatore.aggiornaVelocita(deltaAcc,
                                            this.velocita);
    }

    public int getVelocita() {
        return this.velocita;
    }

     public void setErogatore(IErogatore erogatore) {
         this.erogatore=erogatore;
     }
}
```

Figura 3: Applicazione pratica dello Strategy Pattern

Una volta istanziato un oggetto di tipo *Moto*, bisogna utilizzare il metodo *setErogatore* per utilizzare il tipo di erogatore che si preferisce.
Per esempio una possibile implementazione di tale interfaccia potrebbe essere la seguente.

```
public class ErogatoreDolce implements IErogatore{
    int velMAX=300;
    public int aggiornaVelovita(int deltaAcc, int velocita) {
        int vel=0;
        if(deltaAcc!=0)
            vel=deltaAcc * 20+ velocita;
        else
            vel=velocita+5;
        if(vel>this.velMAX)
            vel=velMAX;
        return vel;
    }
}
```

Seguendo lo stesso approccio possono essere implementati altri tipi di erogatori che implementano l'interfaccia *IErogatore*; nei sorgenti allegati nel CD è possibile reperire le classi *ErogatoreBrusco* ed *ErogatoreMedio*.

STRATEGY PATTERN

A questo punto analizziamo il Class Diagram di questa soluzione.
Il lettore più esperto avrà già capito che questo diagramma è noto al mondo informatico in quanto rispecchia la definizione di un noto pattern: lo *Strategy Pattern*.
Questo pattern consente la definizione di diversi algoritmi (Concrete Strategy) intercambiabili tra loro ed appartenenti alla stessa famiglia (Strategy).
È ovvio che nel nostro caso il ruolo di *Strategy* è ricoperto dall'interfaccia *IErogatore* mentre i *Concrete Strategy* sono le sue implementazioni concrete (es: ErogatoreDolce). Inoltre la definizione del pattern prevede un altro ruolo che è il Context. Esso è rappresentato dalla classe (nel nostro caso è la classe Moto) che si affida allo Strategy per l'esecuzione di un algoritmo.

CONCLUSIONI

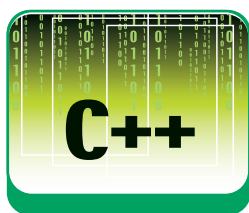
In questo articolo abbiamo studiato i quattro principi che stanno alla base di una buona programmazione Object Oriented. Inoltre abbiamo visto che in alcuni casi la loro corretta applicazione ci porta ad utilizzare i pattern più comuni in modo molto naturale.
Va comunque detto che non sempre è possibile applicare alla lettera tali principi, a volte siamo alle prese con framework o moduli esterni che peccano da questo punto di vista, altre volte sono fattori esterni alla programmazione (scadenze stringenti di progetto, o rilasci troppo ravvicinati) ad impedircelo. Il nostro consiglio è quello di applicare sempre questi principi e di utilizzarli soprattutto nel valutare l'esigenza di refactoring nel nostro software.
L'idea di base è comunque che un'applicazione non sia da considerarsi come statica ma come un elemento "vivo" a cui spesso saremo costretti ad apportare modifiche. Un buon progetto iniziale ed una struttura programmativa orientata alla gestione futura, se da un lato, ci costringerà ad una modalità iniziale più rigida, si rivelerà un elemento di successo per tutte le modifiche che avremo necesstà di apportare in futuro.

Roberto Sidoti

L'AUTORE

Roberto Sidoti è Ingegnere Informatico, in passato si è occupato di servizi VoIP; attualmente lavora come Functional Designer per una multinazionale di consulenza specializzata nello sviluppo di componenti per applicazioni Enterprise.

GESTIAMO IN MODO SICURO LE STRINGHE

NEL 1989 IL COMITATO ANSI AMMISE CHE SE IL C++ NON AVESSE AVUTO UN BUON SUPPORTO NATIVO PER LE STRINGHE SAREBBE "CORSO DEL SANGUE PER LE STRADE". COME E QUANTO FUNZIONANO GLI OGGETTI STRING, OGGI?

Questa seconda parte è dedicata al text processing in C++. Nel primo appuntamento abbiamo visto quali strumenti mette a disposizione il C++ per la creazione e l'estrazione di dati nelle stringhe, considerando anche le soluzioni offerte da varie librerie e sistemi operativi, e terminando con qualche considerazione sull'internazionalizzazione.
Così adesso sappiamo prendere dei dati e inserirli in una stringa senza mandare in crash il sistema. Purtroppo il lavoro del programmatore non si esaurisce qui: con le stringhe bisogna anche lavorarci! Quasi sempre ci si ritrova, ad esempio, a doverle **manipolare** in qualche modo: concatenando, inserendo, rimuovendo o prelevando caratteri o intere stringhe, oppure alterando le maiuscole, o eliminando gli spazi superflui. Possiamo aver bisogno di **effettuare confronti**: verificare, ad esempio, se una stringa contiene o meno un certo carattere o un'intera sottostringa. Oppure possiamo chiederci se due stringhe sono uguali, e in base a quale criterio. Ad esempio: "CIAO" è uguale a "ciao"? Dipende, se si fa o meno differenza fra maiuscole e minuscole.
Infine, un programmatore può trovarsi di fronte alla necessità di **verificare la validità** del contenuto della stringa, controllando se essa aderisce o meno ad uno specifico modello formale. L'utente ha inserito un indirizzo e-mail, nella forma nome@dominio.it? Oppure: la sequenza inserita è un codice fiscale?
E poi ci sono i casi più intricati. Il codice fiscale inserito corrisponde effettivamente ai dati di nascita, o l'utente si è inventato tutto? Per saperlo, occorre andare al di là della semplice struttura formale, interpretandone anche il significato. Per un codice fiscale è piuttosto semplice, dal momento che ha una struttura fissa, ma non è detto che sia sempre così. Un codice può essere formato da più espressioni innestate (quanto vale la stringa "2+2*(5-4)"?), fino ad arrivare ai veri e propri linguaggi (dai tag XML in su). Al momento, il C++ standard offre soluzioni ottimali solo per alcuni di questi problemi, mentre per altri fatica un po', richiedendo al programmatore qualche giro di troppo. I compiti più avanzati, poi, sono completamente fuori portata.

In questa puntata partiremo dalle basi del linguaggio per analizzare le possibilità attuali delle stringhe in C++. Ne sottolineeremo anche i limiti, dedicando tutto resto di questa miniserie a superarli, grazie al supporto di librerie esterne valide e compatibili. Oltre ad essere utilissime, molte soluzioni che vedremo faranno parte integrante di C++09, la prossima versione del C++.

UN APPROCCIO INGENUO ALLE STRINGHE

Non si possono capire le soluzioni avanzate, se prima non si conoscono alla perfezione i meccanismi di base del linguaggio. Sapete com'è definito internamente il tipo **string**? Se la risposta è no, seguite i prossimi paragrafi: vi aiuteranno a capire perché la STL è fatta così, e come funzionano le soluzioni che incontreremo in futuro. Facciamo finta di non conoscere nulla della definizione delle stringhe in C++, e di essere noi a realizzare la libreria standard. Dal momento che una stringa è un **vettore di caratteri**, potremmo definirla così. Che ne dite?

```
typedef string std::vector<char>;
```

Uno dei tanti vantaggi di questa scrittura sta nel fatto che è generica, e permette di definire anche stringhe di "caratteri" che non siano char.

```
typedef wstring std::vector<wchar_t>;
```

È un punto importante: non vogliamo essere accusati di insensibilità verso tutte quelle culture che usano alfabeti e codifiche diversi dal nostro. Ma ciò crea un problema: caratteri diversi richiedono funzioni di valutazione, confronto, ricerca e manipolazione diverse. Per questo potremmo pensare di definire una classe astratta character:

```
class std::character {
public:
    character(int); //costruttore da intero
```

```
    //confronti fra caratteri
    virtual bool operator==(const character& b) = 0;
    virtual bool operator<(const character& b) = 0;
    …
};
```

Character, poi, può essere ereditata dai tipi concreti che definiscono un singolo insieme di caratteri:

```
class std::ASCIIChar : public std::character {…};
class std::UnicodeChar : public std::character {…};
class std::ChineseChar : public std::character {…};
```

E i tipi di caratteri ereditati, infine, possono utilizzati in vettori per creare delle stringhe.

```
typedef std::ASCIIString std::vector<ASCIIChar>;
typedef std::UnicodeString
                                std::vector<UnicodeChar>
    …
```

CHAR TRAITS

Quello mostrato nel paragrafo precedente è un approccio possibile, ma il C++ **non** l'ha seguito, per almeno due buoni motivi. Il primo è l'efficienza: le funzioni virtuali sono più lente e le classi derivate devono memorizzare una vtable. Il secondo è di ordine pratico: dov'è finito il caro e vecchio **char**? Questo tipo di carattere esiste già, ed è quello che viene usato comunemente nella programmazione standard. Lo buttiamo via? L'approccio seguito dal C++, invece, è concettualmente molto simile a quello appena visto, ma fa ricorso alla programmazione generica. Al posto di definire la classe base character, la STL usa il tipo **char_traits**. Vi state chiedendo dove sia la differenza? Sta nel fatto che char_traits è una **classe template**, ed è definita così:

```
template<typename Character> struct char_traits {};
```

Badate che non ho scritto le parentesi graffe per dire: "qui va l'implementazione". char_traits è proprio definita così… una struttura vuota! Evidentemente, c'è sotto un trucco, che peraltro è molto usato nella programmazione C++ generica. Una classe traits, infatti, è un insieme (ma un esperto di metaprogrammazione la definirebbe dispregiativamente un'accozzaglia, o blob) di tipi e funzioni membro statiche che descrivono il tipo passato come **parametro**. Char_traits, quindi, è vuota perché in sé non ha alcun'importanza: quelle che contano sono le sue **specializzazioni**. Per fare un esempio pratico, ecco un estratto di char_traits<char>:

```
//specializzazione di char_traits per i char
template<> struct char_traits<char> {
    //definizione dei tipi
    typedef char char_type;//tipo di carattere
    typedef int int_type;//tipo intero corrispondente
    typedef streampos pos_type;//tipo di offset negli
                                          stream standard
    //… altre definizioni di tipo
    //funzioni di confronto fra caratteri.
    static bool eq(const char_type& a , const char_type&
                                          b); // a == b?
    static bool lt(const char_type& a , const char_type&
                                          b); // a < b?
    //… altre funzioni di confronto
    //funzioni su array. Esempi:
    static size_t length(const char_type* str); // quanto
                                          è lunga str?
    static int compare(const char_type* str1, const
                  char_type* str2); //confronta due stringhe
    …
};
```

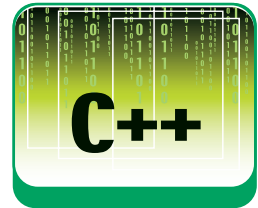
Come vedete, in questo estratto si definiscono dei tipi e delle funzioni membro statiche. Se char_traits è l'equivalente generico della classe base character che abbiamo visto sopra, char_traits<char> è l'equivalente di ASCIICharacter. Tutte le altre specializzazioni di char_traits per altri tipi di caratteri hanno lo stesso aspetto. Ad esempio:

```
//specializzazione di char_traits per i wchar_t
template<> struct char_traits<wchar_t> {
    //definizione dei tipi
    typedef char char_type;     //tipo di carattere
    typedef int int_type;//tipo interocorrispondente
    …
    //funzioni di confronto fra caratteri
    static bool eq(const char_type& a , const
                                  char_type& b); //a == b?
    …
};
```

E così via.

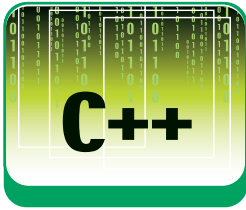
BASIC_STRING

Grazie a char_traits, possono essere scritte funzioni e classi che si comportino diversamente a seconda del parametro passato. Siete di fronte a ciò che in gergo viene chiamato **polimorfismo statico**. Per sfruttarlo, occorre abbandonare l'idea di un vector<char> e creare un nuovo contenitore che faccia riferimento internamente a char_traits. La STL lo chiama **basic_string**, e lo definisce così:

```
template<
typename Character,
typename Traits = char_traits<Character>,
typename Allocator = allocator<Character>
```

"INSTALLARE" BOOST

Per usare ciò che è spiegato in quest'articolo, non serve alcuna "installazione" particolare di boost. Basta scaricare i sorgenti da www.boost.org, e impostare le directory di inclusione nel proprio IDE (se ne si sta usando uno).

"INSTALLARE" STRING_ALGO

Per usare gli string algorithm non è necessario collegare alcuna libreria. E' sufficiente avere impostato boost fra le directory d'inclusione, e includere il file "<boost/algorithms/string.hpp>". Per evitare di dover specificare sempre boost:: davanti a ogni funzione, è bene dichiarare, da qualche parte prima dell'uso, using namespace boost;

NOTA

METAPROGRAMMAZIONE

Se la spiegazione del funzionamento di char_traits ha illuminato la vostra mente affacciandola a nuove vette di perversione, forse avete la stoffa dei metaprogrammatori template e non lo sapete ancora. Questa rubrica tratterà alcune tecniche di metaprogrammazione fra diversi numeri... nel frattempo potete leggere l'indispensabile "C++ Template Metaprogramming" di Dave Abrahams e Aleksey Gurtovoy.

LEZIONI D'INCAPSULAMENTO

Se tutta quella storia sul "Preferire funzioni non membro non friend alle funzioni membro" non vi convince molto, è decisamente il caso di leggervi il Tema numero 23 di Effective C++ (3ed) di Scott Meyers.

```
> class std::basic_string;
```

Come vedete, il programmatore deve soltanto specificare il tipo di carattere (indicato qui con Character) e, se esiste una specializzazione char_traits <Character>, basic_string sarà automaticamente in grado di accedere alle funzioni che abbiamo visto nel precedente paragrafo. Inoltre, se si vuole, è possibile specificare un char_traits diverso e personalizzato (come vedremo fra poco). Per velocizzare le cose, la STL definisce anche i seguenti typedef:

```
typedef std::string std::basic_string<char>
typedef std::wstring std::basic_string<wchar_t>
```

PERSONALIZZARE CHAR_TRAITS

È importante sapere che basic_string si basa, nelle sue operazioni fondamentali, su char_traits. Prendiamo il caso dell'operatore di confronto.

```
std::string stringa1 = "Ciao", stringa2 = "CIAO";
std::cout << (stringa1 == stringa2); //Vero? Falso?
```

Il codice qui sopra riprende la domanda posta nell'introduzione: "Ciao" e "CIAO" sono uguali? Basic_string non risponde direttamente, ma lo chiede a char_traits. Per la specializzazione char_traits<char> della libreria standard la risposta è **no**. Il confronto è un'operazione case sensitive: fa differenza fra maiuscole e minuscole.

E se volessimo una stringa che ***non*** fa differenza fra maiuscole e minuscole? Basterebbe implementare un char_traits appropriato:

```
struct char_traits_indulgente : char_traits<char> {
   static bool eq(char c1, char c2) {
      return toupper(c1) == toupper(c2);
   }
   static int compare(const char* s1, const char* s2,
                                             size_t n) {
      return memicmp(s1, s2, n);
   }
};
```

Il nuovo ***char_traits_indulgente*** deriva direttamente da char_traits, ridefinendo le funzioni di confronto fra caratteri (eq) e fra stringhe (compare). In quest'ultima, usiamo la funzione memicmp, che non è standard ma è molto diffusa fra i migliori compilatori. Se seguite questa rubrica da un po', forse vi ricorderete che abbiamo fatto qualcosa del genere qualche numero fa, quando abbiamo definito un allocatore che usasse malloc derivando da allocator.

Anche in questo caso ripeto lo stesso monito dell'altra volta: derivare da classi concrete prive di distruttore virtuale **è Male**, ma in questi casi è consentito, dato che non abbiamo certo intenzione di usare char_traits_indulgente in modo polimorfico.

Ora possiamo usare char_traits_indulgente per creare una ***stringa_indulgente***, così:

```
typedef basic_string<char,
                char_traits_indulgente> stringa_indulgente;
```

E il gioco è fatto. La nuova stringa userà le funzioni di confronto "indulgenti", che non fanno differenza fra maiuscole e minuscole:

```
stringa_indulgente stringa1 = "ciao", stringa2 =
                                                  "CIAO";
std::cout << (stringa1 == stringa2); //vero!
```

BASIC_STRING COME CONTENITORE STL

Basic_string definisce costruttori, distruttore, operatore d'assegnamento, di uguaglianza, e di confronto. Inoltre, offre alcune funzioni membro ottimizzate per la gestione della stringa. Non bisogna mai dimenticare che basic_string offre tutte le possibilità di un contenitore STL: le funzioni ***push_back*** e ***append***(), ad esempio, inseriscono degli elementi in coda. ***Insert*** inserisce dei caratteri in mezzo, **erase** li rimuove, e ***clear*** svuota tutto il buffer. ***Size*** (o length) restituisce il numero di caratteri della stringa e **capacity** la reale dimensione corrente del buffer. **Resize** aumenta le dimensioni del buffer aggiungendo un carattere alla fine. E, ovviamente, [] e at possono essere usati per l'accesso casuale.

```
string str("CIAO…");  //costruttore da char[]
str.erase(4, 3);        //str = "CIAO"
str += "MONDO";       //str = "CIAOMONDO"
str.push_back('!');    //str = "CIAOMONDO!"
str.insert(4, ", ");     //str = "CIAO, MONDO!"
str.resize(14, '!');      //str = "CIAO, MONDO!!!"
cout << str.length();               //14
cout << str.capacity();             //14 o più
cout << str[15];      //comportamento indefinito
cout << str.at(15); //eccezione out_of_range
```

BASIC_STRING E GLI ALGORITMI STL

In quanto contenitore STL a tutti gli effetti, basic_string permette di accedere agli iteratori, e quindi è possibile utilizzare su un basic_string tutta la vasta gamma degli algoritmi previsti dalla libreria <algorithm> del C++:

```
#include <iostream>
```

```
#include <string>
#include <algorithm>

using namespace std;

bool SonoAnagrammi(const string& parola1, const
                                  string& parola2) {
   string firma1(parola1), firma2(parola2);

   sort(firma1.begin(), firma1.end());
   sort(firma2.begin(), firma2.end());

   return (firma1 == firma2);
}

int main(int argc, char **argv) {
   if (argc < 3)
      return EXIT_FAILURE;

   if (SonoAnagrammi(argv[1], argv[2]))
      cout << "I due termini sono
                                  anagrammi.\n";
   else
      cout << "Riprova. Sarai piu' fortunato.\n";

   return EXIT_SUCCESS;
}
```

L'esempio qui sopra mostra un piccolo programmino che verifica se due argomenti passati sono o meno l'anagramma l'uno dell'altro, per mezzo dell'algoritmo sort. Sort ordina i caratteri in modo crescente, in modo da ottenere una "firma" di confronto. Potete ben immaginare che questo è probabilmente l'unico caso al mondo in cui avrà mai senso "ordinare" i caratteri di una stringa. È emblematico, purtroppo: anche se funzionano benissimo, **spesso gli algoritmi STL non hanno molto senso se applicati alle stringhe**. E anche quando hanno senso, spesso non sono molto ottimizzati. E anche quando sono ottimizzati, spesso hanno un aspetto molto "innaturale". (mi fermo qui.) (potrei continuare.) Per questo, basic_string ha delle funzioni membro specifiche per implementare alcune operazioni comuni per i testi.

CONVERSIONI A STRINGHE C-LIKE

Fra le necessità specifiche di basic_string, c'è la conversione dei dati. Alcune librerie di stampo C, infatti, non "capiscono" le stringhe C++, e hanno bisogno di un rozzo array di char. Basic_string implementa ben tre funzioni, per queste situazioni: **data** di accedere ad un array di caratteri, **c_str** una stringa "in stile C" a terminatore nullo, e **copy** copia l'array in un buffer già creato (senza terminatore).

Tipicamente si usa c_str per chiamare le funzioni di libreria C, e copy per evitare di sprecare memoria inutilmente, nel caso si disponga di un array già creato. La funzione data è notoriamente poco utile, come mostra il seguente esempio:

```
   string str("Ciao, mondo!");
   char stringa[20]; //buffer già esistente
   size_t lenStringa = str.copy(stringa, 5); //copia
                        i primi 5 caratteri di str in stringa
   stringa[lenStringa] = '\0';    //aggiunge
                                   terminatore alla fine
   cout << stringa;    //Ciao
   //stringa C-like
   cout << strlen(str.c_str());    //12
   //stringa non terminata
   cout << str.data();    //crash! :)
```

FUNZIONI DI RICERCA DI BASIC_STRING

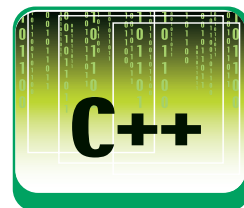
Infine, basic_string permette di reperire sequenze di caratteri all'interno della stringa – ovverosia delle sottostringhe. La funzione substr(partenza, numerocaratteri) è fatta apposta:

```
string str("Ciao, mondo!");
cout << str.substr(6, 5);
   string str("www.ioprogrammo.it");
   cout << str.find("."); //3
   cout << str.rfind("."); //15
   cout << str.find_first_not_of("w."); //4
   cout << str.find_last_of("z"); //npos
```

L'ultima istruzione è interessante. Si richiede al C++ di ricercare l'ultima zeta presente nella stringa str. Solo che in str… non c'è nessuna zeta! In questo caso, il valore restituito sarà la costante ***npos*** (*no position*). Npos è impostato sul valore limite per il buffer di una stringa (-1, dato che un size_t è senza segno), e pertanto usarlo come indice è un errore, che genera un'eccezione di tipo ***range_error***.

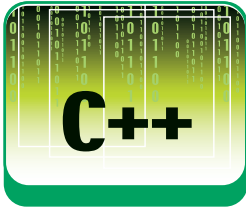
```
   string str("www.ioprogrammo.it");
   cout << str[str.find_first_not_of("w.")];   //'i'
   cout << str[str.find_last_of("z")];
```

Una volta reperita una sottostringa, è possibile sostituirla con un'altra, oppure rimuoverla del tutto (il che, alla fin fine, è un caso particolare di sostituzione con una stringa vuota). Replace(inizio, numerocaratteri, sostituzione) permette di farlo.

```
   string str("www.ioprogrammo.it");
   size_t inizio = str.find(".") + 1;
   size_t fine = str.rfind(".");
   size_t n = fine - inizio;
```

STRING E STL

Per tutti i dettagli su string e gli algoritmi STL, consultate un buon testo di riferimento. Il mio personalissimo preferito è "The C++ Standard Library, a tutorial and reference" di Nicolai Josuttis.

CASE (IN) SENSITIVITY

La creazione di un char_traits case insensitive è trattata in varie fonti, e una delle migliori è senz'altro "Exceptional C++" di Herb Sutter. Nel Tema numero 2, si affronta la questione in profondità, e si risponde anche la domanda "è sicuro derivare da classi concrete, se non si intende usare il polimorfismo dinamico? [sì]", sulla base del Principio di Sostituzione di Liskov Generico.

NOTA

BOOST?

Un'ottima introduzione a Boost è il testo di Bjorn Karlssonn: "Beyond the C++ Standard Library". Nontratta alcuni argomenti, come gli string_algo visti in questa puntata, ma fornisce informazioni in abbondanza su boost::regex, boost::range, e altri meccanismi fondamentali.

QUALI STRINGHE?

Se ci fate caso, la maggior parte degli algoritmi di string_algo presentati qui non richiede necessariamente che i parametri passati siano di tipo basic_string. Qualsiasi sequenza permetta di ottenere degli iteratori va bene: dagli array di char ai contenitori più esotici.

```
str.replace(inizio, n, "robertoallegra");
cout << str; //www.robertoallegra.it
```

I LIMITI DI BASIC_STRING E BOOST::STRING_ALGO

Più o meno questo è tutto ciò che offre basic_string. È un punto di partenza, ma soddisfa molto poco le esigenze del programmatore medio che abbiamo riassunto a inizio paragrafo. Manca del tutto il supporto per alcune operazioni fondamentali, come la conversione in maiuscolo o in minuscolo, l'eliminazione degli spazi, la ricerca avanzata di sottostringhe. È possibile costruire facilmente delle funzioni che si occupino di farlo, ed è per questo motivo che è giusto che queste **non** siano inglobate in basic_string, seguendo l'inviolabile Regola Di Stile che vuole che si preferiscano le funzioni non membro non friend definite esternamente alla classe, alle funzioni membro. Il problema è che queste funzioni non membro non friend eccetera… non ci sono! Che state facendo? Aspettate un attimo, prima di buttarvi sulla tastiera a scriverne un'implementazione fai-da-te… Pavol Droba si è buttato sulla tastiera prima di noi, e ha scritto un'implementazione di tutti gli algoritmi relativi alle stringhe che mancavano nella STL. L'ha presentata alla comunità di Boost, questa l'ha passata al setaccio, e infine l'ha adottata, inserendola nell'header <boost/algorithm/string.hpp>. L'implementazione interna di queste funzioni è piuttosto complessa, e utilizza trucchi di metaprogrammazione tipici di Boost per semplificare la vita al programmatore finale. Nel resto di quest'articolo vedremo alcune delle possibilità offerte da questi algoritmi, rimandando ai prossimi appuntamenti (e alla documentazione di boost) per quelli più avanzati.

MAIUSCOLO, MINUSCOLO, E BOOST::RANGE

Uno dei buchi più sorprendenti della STL è la mancanza di un algoritmo per la conversione dal minuscolo al maiuscolo e viceversa. È vero che il C++ definisce in <locale> le funzioni toupper e tolower, ma queste lavorano solo su singoli caratteri! Potremmo scrivere la funzione da noi senza scomodare boost, ricordandoci che una stringa è un contenitore STL a tutti gli effetti:

```
#include <locale>
#include <algorithm>
template<typename Iterator>
    inline void to_upper(Iterator begin, Iterator  end) {
        transform(begin, end, begin, toupper);
    }
```

Questa funzione svolge onestamente il suo lavoro, e infatti possiamo usarla facilmente:

```
string str = "www.ioprogrammo.it";
to_upper(str.begin(), str.end());
cout << str;          //WWW.IOPROGRAMMO.IT
```

Internamente, la funzione boost::to_upper (così come boost::to_lower) è implementata più o meno allo stesso modo, ma rende le cose più semplici al programmatore, grazie ad una magia chiamata **boost::Range**. Grazie a Range è possibile passare semplicemente il contenitore, al posto dei due iteratori – che nel 99% dei casi sono i soliti begined end).

```
#include <boost/algorithm/string.hpp>
string str = "www.ioprogrammo.it";
boost::to_upper(str);
cout << str;          //WWW.IOPROGRAMMO.IT
```

Se ci si trova nel restante 1% dei casi, e si ha un effettivo bisogno di indicare una parte della stringa, si può sempre creare un iterator_range, passando i due estremi come argomenti del suo costruttore:

```
string str("www.ioprogrammo.it");
size_t inizio = str.find(".") + 1;
size_t fine = str.rfind(".");
iterator_range<string::iterator> ioprogrammo(
    str.begin() + inizio,
    str.begin() + fine
);
boost::to_upper(ioprogrammo);
cout << str;    //www.IOPROGRAMMO.it
```

L'iterator_range va costruito prima della chiamata, e per questo la scrittura può rivelarsi un po' prolissa (ma più chiara). Ma in quanti casi avrete bisogno di rendere maiuscola solo una porzione della stringa?

ALGORITMI PER COPIA E STD::BACK_INSERTER

Molti algoritmi esposti dalla libreria boost::string_algo sono forniti almeno in due forme: in place e per copia. La cosa non è affatto nuova, dal momento che anche tanti algoritmi STL esistono in due o più forme. Si pensi ad esempio a std::replace, che sostituisce gli elementi compresi fra due iteratori. Esiste anche std::replace_copy, che non tocca l'originale, ma crea una sequenza con gli elementi già sostituiti. La libreria boost segue la stessa convenzione: esiste boost::to_upper, che converte una stringa in maiuscolo, ed esiste **boost::to_upper_copy**, che copia la stringa passata in una nuova, trasformata in maiuscolo. A dirla tutta, esi-

stono ben due versioni di to_upper_copy, a seconda delle esigenze. La prima è quella facile, che prende una stringa e ne restituisce una copia trasformata:

```
string str("www.ioprogrammo.it");
string str2(to_upper_copy(str));
cout << str2; //WWW.IOPROGRAMMO.IT
```

Poi c'è il caso più complicato, analogo a quell'1% che abbiamo visto nel paragrafo precedente, in cui si desidera effettuare una copia di un raggio ridotto di una stringa di partenza. In questi casi, si usa la versione to_upper_copy(IteratoreCopia, Range Originale), laddove IteratoreCopia è tipicamente un **inseritore**. L'uso e il funzionamento degli inseritori è noto a chiunque abbia un po' di esperienza nella programmazione per algoritmi C++. Se non siete pratici, potete vedere **back_inserter**(stringa) come una funzione magica che restituisce un iteratore ancor più magico, capace di inserire elementi direttamente in *stringa*.

```
string str("www.ioprogrammo.it");
string str2;
size_t inizio = str.find(".") + 1;
boost::to_upper_copy(back_inserter(str2),
                                   ioprogrammo);
cout << str2;  // WWW.IOPROGRAMMO.IT
```

È una procedura un po' prolissa, ma quante volte vorrete creare una nuova stringa a partire da una porzione di quella originale?

TRIMMING

Un'altra mancanza della STL è una funzione specificamente destinata al trimming. In un mondo perfetto non ci sarebbe bisogno di eliminare gli spazi superflui, ma il nostro è decisamente imperfetto: per colpa dell'utente o del sistema (o di un'associazione a delinquere fra i due) capita spesso di trovare delle povere stringhe strette, da un lato o da entrambi, fra la morsa di una comitiva di spazi inutili.
È certamente possibile organizzare una funzione di trimming usando find_first_not_of (per trovare il primo carattere sinistro che non sia uno spazio) e find_last_not_of (per il lato destro), ma non ce n'è bisogno. Boost::String_algo si occupa anche di questo, mediante le funzioni **trim**, **trim_left**, e **trim_right**. Tutti i sistemi già visti per to_upper e to_lower si applicano anche alle funzioni di trimming: i range, le funzioni di copia, gli inseritori.

```
string str("  www.ioprogrammo.it  ");
  string str2(trim_left_copy(str));   //str2 =
                          "www.ioprogrammo.it  "
string str3(trim_right_copy(str)); //str3 = "
                          www.ioprogrammo.it"
str.trim();          //str = "www.ioprogrammo.it"
```

A volte, a dare fastidio non sono soltanto gli spazi, ma anche altri caratteri: tabulazioni, punti, asterischi, zeri, e così via. Le funzioni di trimming **trim_left_if**, **trim_right_if trim_if** di specificare un predicato per l'identificazione dei caratteri da tagliare:

```
string str("../../../ioprogrammo/");
  string str2 = "C:/" + trim_copy_if(str, is_any_of("./"));
// cout << str2;
```

Nel codice qui sopra abbiamo un indirizzo relativo pieno di punti e slash, e vogliamo trasformarlo in uno assoluto Indichiamo quindi di eliminare ogni carattere '.' o '/' che compaia agli estremi della stringa.

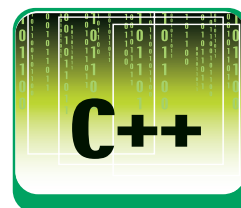
PREDICATI E BOOST::ALL

Ma da dove è saltata fuori la funzione is_any_ofdell'ultimo esempio? È un predicato. Niente di nuovo sotto il sole: la STL è piena di algoritmi_if che richiedono predicati. Is_any_of è uno dei tanti messi a disposizione da boost::string_algo, e restituisce *true* un carattere corrisponde a quelli indicati. I predicati vengono usati dalle funzioni_if come *trim*, o come ***all***, che verifica se un dato predicato è soddisfatto per ogni carattere della stringa:

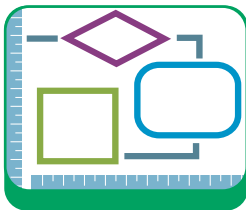
```
//contiene solo cifre?
cout << all("IoProgrammo 114", is_digit()); //falso
cout << all("114", is_digit()); //vero
cout << all("IoProgrammo 114", is_alpha()); //falso
cout << all("IoProgrammo", is_alpha()); //vero
cout << all("accada", is_from_range('a', 'd')); //vero
cout << all("avvenga", is_from_range('a', 'd')); //falso
```

CONCLUSIONI

Abbiamo cominciato questo articolo dalle fondamenta del linguaggio e ci ritroviamo già ad utilizzare una libreria avanzata come boost::string_algo.
Molte necessità devono ancora essere soddisfatte, e molte soluzioni devono ancora essere rivelate. Appuntamento al mese prossimo, in cui faremo un salto in avanti nella libreria string_algo (e in altre più avanzate), scoprendo strumenti ancor più sofisticati.

Roberto Allegra

Il sito www.robertoallegra.it contiene l'elenco degli articoli pubblicati in questa rubrica, con gli inevitabili approfondimenti ed errata corrige. L'e-mail dell'autore è articoli@robertoallegra.it.

L'ADAPTER: METTE TUTTI D'ACCORDO

I MIGLIORI PROGRAMMATORI LAVORANO SPESSO ASSEMBLANDO LIBRERIE E FRAMMENTI DI CODICE SCARICATI DA INTERNET. DI SOLITO QUESTI PEZZI NON SONO PENSATI PER FUNZIONARE INSIEME. MA PER FORTUNA, C'È MODO DI RISOLVERE IL PROBLEMA.

E venerdì. E Salvatore, il preoccupato programmatore, sa che il suo fine settimana è in pericolo. Questa mattina il capo è apparso, silenzioso come un ninja, per chiedere alcune innocenti modifiche all'applicazione gestionale per il Club dei Ricchi cittadino. Il responsabile commerciale ha già venduto le modifiche senza consultare i tecnici, e ha promesso che il tutto sarebbe stato pronto entro lunedì. Salvatore si mette al lavoro sospirando. Aveva ragione sua madre: avrebbe dovuto fare l'avvocato.

IL CLUB DEI RICCHI

Il Club dei Ricchi si basa su un semplice principio: ciascun nuovo socio deve essere presentato da un socio già esistente, che da quel momento in poi diventa il suo responsabile. Se un socio si comporta in modo sconveniente (ad esempio sbevazza il contenuto dello sciacquadita, o racconta una barzelletta da camionisti durante il ricevimento della Contessa Serbelloni), il suo responsabile subisce una sanzione disciplinare.
L'azienda di Salvatore, il preoccupato programmatore, ha sviluppato per il club una semplice applicazione in Ruby. Ecco la classe che identifica un socio:

```
class Member
  attr_reader :name, :surname, :introduced_by

  def initialize(name, surname, introduced_by)
    @name = name
    @surname = surname
    @introduced_by = introduced_by
  end
end
```

Anche se non conoscete Ruby, non dovrebbe essere difficile capire questo codice. Ignorate per il momento la riga che inizia con *attr_reader*. Il metodo *initialize()* è l'equivalente Ruby di un costruttore. Accetta tre argomenti: il nome del socio, il suo cognome e la persona che l'ha presentato, che sarà a sua volta un *Member*. Se quest'ultimo argomento ha un valore nullo *(nil)*, allora il socio non ha un responsabile, ed è dunque il fondatore del club.
I tre argomenti di *initialize()* vengono conservati in altrettante variabili di istanza (Ruby le chiama "variabili di istanza", ma se preferite potete chiamarle "campi"). In Ruby, le variabili di istanza si riconoscono perché hanno una "@" come prefisso. Come per le normali variabili, non c'è bisogno di dichiarare le variabili di istanza: nascono immediatamente la prima volta che gli si assegna un valore.
In conclusione, posso creare un fondatore così:

```
piergigi = Member.new("Piergigi", "Rossi", nil)
```

Quando chiamo *new()* sulla classe *Member*, Ruby crea un'istanza della classe e la inizializza chiamando *initialize()*. In questo caso abbiamo creato un socio fondatore assegnando un valore nullo a *introduced_by*. Tipicamente, dobbiamo dare un responsabile al nuovo socio:

```
pieruga = Member.new("Pieruga", "Verdi", gino)
```

Le variabili di istanza in Ruby sono sempre private. Quindi non posso leggerle direttamente:

```
piergigi.@surname  # Errore!
```

La riga della classe *Member* che inizia con *attr_reader* definisce però tre proprietà in sola lettura che hanno gli stessi nomi delle tre variabili di istanza (senza il prefisso "@"), e le lega automaticamente alle variabili di istanza. Quindi posso scrivere:

```
puts piergigi.surname
```

puts è l'istruzione per stampare sullo schermo. Questa riga di codice stampa:

```
Rossi
```

Conoscenze richieste

Software

Impegno

Tempo di realizzazione

NUOVI RICCHI

Grazie alla recente speculazione edilizia, il club ha ricevuto un costante influsso di nuovi membri. Con tanta gente in giro, non è sempre facile capire chi sia responsabile di chi. Quindi i gestori del club ci hanno chiesto di implementare una visualizzazione ad albero che mostra tutti i soci. Ciascun nodo dell'albero è un socio, e i suoi "figli" sono i soci di cui è responsabile. La radice è il fondatore del Club, che non ha alcun responsabile. Ad esempio:

```
Giangigi Rossi
  Pieruga Verdi
    Piermalloppo Gialli
    Filiberto Magenti
  Giorgiarmando Bianchi
```

Questa visualizzazione indica che il fondatore Rossi è responsabile di Verdi e Bianchi. A sua volta, Verdi è responsabile di Gialli e Magenti. Per complicare le cose, la visualizzazione ad albero deve funzionare su almeno tre interfacce diverse: una semplice visualizzazione testuale come quella qui sopra, una visualizzazione interattiva simile a quella di un file system, e una pagina web.
È impossibile pensare di scrivere tutta questa roba entro oggi. Ma Salvatore, l'eroico programmatore, non si perde d'animo: gli basta una breve ricerca su Internet per trovare una classe di nome *TreePanel* che genera automaticamente tutte le interfacce "ad albero" che gli servono.

```
class TreePanel
  def initialize(root)
    raise "Not a root" if root.parent
    @nodes = [root]
  end

  def add(node)
    raise "No such parent" if !@nodes.include?
                                          node.parent
    @nodes << node
  end

  def show
    show_subtree(@nodes[0])
  end

private

  def show_subtree(node, depth = 0)
    puts " " * depth + node.to_s
    children_of(node).each {|child| show_subtree(child,
                                         depth + 1) }
  end

  def children_of(node)
    @nodes.find_all {|n| n.parent == node }
```

```
  end

  # seguono i metodi per visualizzare l'albero su
  # una pagina web, eccetera
end
```

Salvatore sa che non ha bisogno di entrare nei dettagli di questa classe. Si tratta di un solido prodotto open source, e gli serve solo capire come usarla. Il nostro baldo programmatore esamina rapidamente i metodi principali di *TreePanel*:

```
def initialize(root)
  raise "Not a root" if root.parent
  @nodes = [root]
end
```

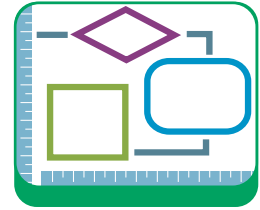
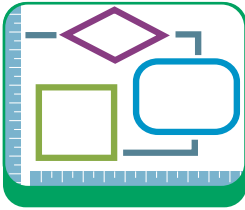
Il "costruttore" di *TreePanel* riceve un nodo che diventa la radice dell'albero. Questo codice chiede al nodo quale chi sia suo padre leggendo la proprietà *parent*. Se *parent* esiste, cioè se non è *nil*, allora il nodo non può essere usato come radice, e il codice solleva un'eccezione (in Ruby, come in Perl, si può mettere l'*if* dopo la condizione anziché prima). Il metodo *initialize()* definisce anche una variabile di istanza *@nodes*, un array che conterrà tutti i nodi dell'albero, con la radice in prima posizione.

```
def add(node)
  raise "No such parent" if !@nodes.include? node.parent
  @nodes << node
end
```

Questo metodo aggiunge un nodo all'albero. Se qualcuno cerca di aggiungere un nodo il cui padre non è già nella lista dei nodi registrati, la prima riga lancia un'eccezione (un nuovo socio deve essere presentato da qualcuno che è già socio, non da uno sconosciuto). La seconda riga aggiunga il nuovo nodo all'array dei nodi.

```
def show
  show_subtree(@nodes[0])
end
```

show() chiama un metodo privato, *show_subtree()*, che stampa un intero sotto-albero a partire da un singolo nodo. Salvatore non si preoccupa di capire bene come funziona *show_subtree()*. Gli basta sapere che show() passa a *show_subtree()* il nodo radice, stampando così l'intero albero.
Per concludere, Salvatore dà uno sguardo al metodo privato *children_of, usato da show_subtree*:

```
def children_of(node)
  @nodes.find_all {|n| n.parent == node }
end
```

Questo metodo usa qualche magico trucco Ruby per trovare tutti i figli del nodo dato, esaminando tutti i nodi dell'albero e chiedendo a ciascuno chi sia suo padre. Questo è sicuramente un modo inefficiente per trovare i figli di un nodo, ma per un piccolo albero è accettabile. Questa classe manipola degli oggetti che sono nodi dell'albero, e vuole che questi oggetti abbiano una determinata interfaccia. In particolare:

- *initialize(), add() e children_of()* vogliono che ciascun nodo abbia una proprietà *parent* che rappresenta il suo nodo padre.
- *show_subtree()* stampa ciascun nodo sullo schermo. In Ruby, questo significa che ciascun nodo deve avere un metodo *to_s()* che restituisce la sua rappresentazione come stringa.

Salvatore sa che il modo migliore per capire un pezzo di codice è usarlo, quindi scrive una classe di test che soddisfa queste due condizioni e finge di essere una directory sul disco:

```
class Directory
  attr_accessor :parent

  def initialize(name)
    @name = name
  end

  def to_s
    @name
  end
end
```

attr_accessor funziona come attr_reader, ma definisce una proprietà di lettura e scrittura. Questo significa che non solo posso leggere il parent di un nodo, ma posso anche assegnargli un valore:

```
figlio.parent = padre
```

Questo soddisfa la prima condizione attesa da TreePanel. Inoltre Node accetta un nome nel suo costruttore, e questo nome è lo stesso che viene restituito dal metodo di conversione in stringa to_s(). Questo soddisfa la seconda condizione. Il test di Salvatore usa questa classe per testare il TreePanel costruendo un finto albero di directory:

```
root = Directory.new("c:")
downloadFolder = Directory.new("download")
musicFolder = Directory.new("music")
videosFolder = Directory.new("videos")
```

Dopo aver creato i nodi dell'albero, il test li lega insieme:

```
downloadFolder.parent = root
musicFolder.parent = downloadFolder
videosFolder.parent = downloadFolder
```

Infine, il test costruisce il TreePanel e lo mostra sullo schermo:

```
panel = TreePanel.new(root)
panel.add downloadFolder
panel.add musicFolder
panel.add videosFolder
panel.show
```

Notate che le parentesi nelle chiamate ai metodi sono opzionali in Ruby, e Salvatore ha usato questa caratteristica del linguaggio per risparmiare un po' di codice. Ecco il risultato:

```
c:
  download
    music
    videos
```

Sembra che la classe funzioni bene. Salvatore deve solo integrarla nel sistema. A questo punto, però, sorge un problema.

DUE INTERFACCE E DUE MISURE

Il sistema gestionale del Club dei Ricchi tratta oggetti di tipo *Member*. Ma questi oggetti non soddisfano le condizioni attese dal *TreePanel*: non hanno un parent, né un *to_s()*. Quindi non posso usare un Member come nodo di un TreePanel.
Si potrebbe risolvere il problema aggiungendo le caratteristiche mancanti alla classe *Member*. Ma se aggiungere un metodo *to_s()* sembra un'idea sensata, aggiungere una proprietà *parent* vuol dire inquinare il codice di *Member*. Un *Member* deve essere usato in molte circostanze diverse, e non è il caso che questa classe sappia di essere anche un nodo dell'albero.
In alternativa si potrebbe modificare *TreePanel* perché legga la proprietà *introduced_by* anziché parent, ma anche in questo caso si tratterebbe di un brutto hack. La classe *TreePanel* dovrebbe trattare qualsiasi nodo, non solo i *Member*. Inoltre, diventerebbe difficile aggiornare *TreePanel* quando il suo autore ne rilascerà una nuova versione. Il fatto è che *TreePanel* e *Member* vivono su piani diversi: la prima classe fa parte della presentazione, la seconda del modello business. Non avrebbe senso cablare una delle due classi per andare incontro alle esigenze dell'altra.
Quello che ci serve è un traduttore, qualcuno che

renda comprensibili i *Member* al *TreePanel*. Questo è il lavoro del pattern Adapter.

IL PATTERN ADAPTER

Pensate a quante volte, quando volete collegare un dispositivo elettrico alla presa, vi accorgete che lo standard della presa nel muro è diverso da quello della spina. Quello che si fa in questi casi è cercare un adattatore.
La stessa cosa succede con il codice: il client parla ad una certa interfaccia, ma gli oggetti che vogliamo passargli hanno un'interfaccia diversa. Anche in questo caso ci serve un oggetto che faccia da adattatore. Il client parla all'adattatore, che traduce le chiamate verso l'oggetto adattato.
Salvatore, l'astuto programmatore, scrive quindi una classe *MemberNode* con due caratteristiche:

- *MemberNode* si comporta come un nodo, in modo che *TreePanel* possa usarlo tranquillamente. Questo significa che *MemberNode* ha una proprietà parent e un metodo *to_s()*.
- *MemberNode* contiene un *Member*, e delega a lui tutte le operazioni che riguardano il modello.

```
class MemberNode
  def initialize(member)
    @member = member
  end

  def parent
    @member.introduced_by
  end

  def to_s
    @member.name + " " + @member.surname
  end
end
```

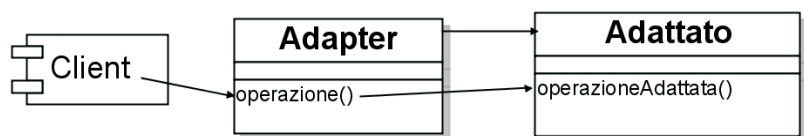
MemberNode incapsula un *Member* che gli viene passato in costruzione e lo trasforma in un oggetto manipolabile da *TreePanel*. Come la classe di test Node, anche *MemberNode* ha una proprietà di nome parent, il cui valore è lo stesso della proprietà *introduced_by* del *Member* incapsulato. In realtà parent è un metodo, ma in Ruby non ci sono vere differenze tra proprietà e metodi: posso chiamare parent senza usare le parentesi, che è quello che succede nel codice di *TreePanel*.
MemberNode ha anche un metodo to_s() che trasforma l'oggetto in una stringa. Questo metodo concatena semplicemente il nome e il cognome del *Member* incapsulato.
Ora possiamo scrivere un test che mette insieme il client (*TreePanel*), l'Adapter (*MemberNode*) e l'oggetto Adattato (*Member*). Dobbiamo creare ciascun nodo e "avvolgerlo" in un Adapter:

```
giangigi = MemberNode.new(Member.new("Giangigi",
                                      "Rossi", nil))
pieruga = MemberNode.new(Member.new("Pieruga",
                                     "Verdi", giangigi))

giorgiarmando =
        MemberNode.new(Member.new("Giorgiarmando",
                                   "Bianchi", giangigi))
piermalloppo =
    MemberNode.new(Member.new("Piermalloppo", "Gialli",
                               pieruga))
filiberto = MemberNode.new(Member.new("Filiberto",
                                       "Magenti", pieruga))
```

Ora possiamo creare il *TreePanel*, aggiungergli i nodi "adattati" e stampare l'albero:

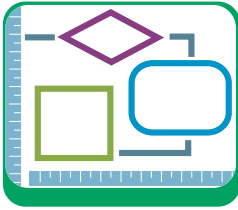
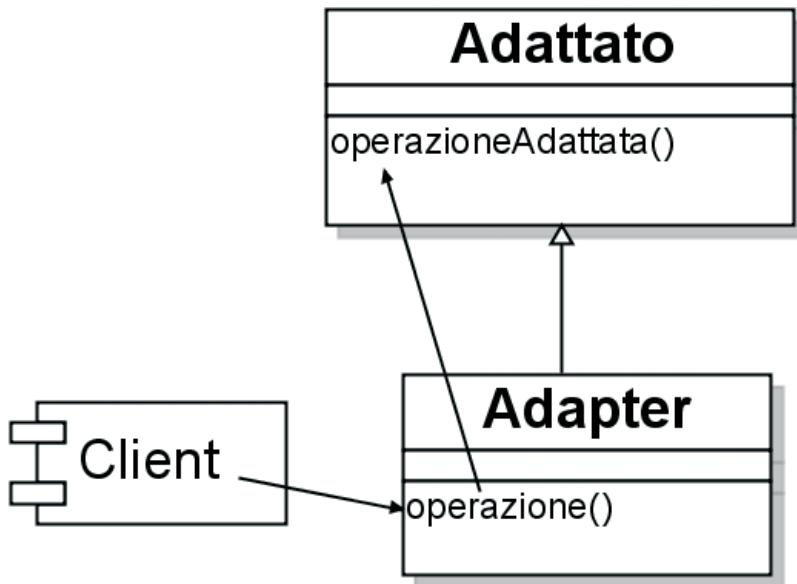
```
panel = TreePanel.new(giangigi)
panel.add pieruga
panel.add giorgiarmando

panel.add piermalloppo
panel.add filiberto
panel.show
```

Il risultato è:

```
Giangigi Rossi
  Pieruga Verdi
    Piermalloppo Gialli
    Filiberto Magenti
  Giorgiarmando Bianchi
```

Funziona!

VARIAZIONI SUL TEMA

MemberNode è una classe completamente separata da *Member*. Però incapsula un *Member*, e delega a lui la maggiore quantità possibile di operazioni. Quindi il legame tra un *MemberNode* e un *Member* avviene a livello di *oggetti*. Per questo motivo, questa versione del pattern si chiama anche Object Adapter.
Esiste una variante dello stesso pattern in cui la classe dell'Adapter eredita dalla classe dell'Adattato:

```
class MemberNode2 < Member
```

```
def parent
  introduced_by
end
def to_s
  name + " " + surname
end
end
```

Dato che un *MemberNode2* è un *Member*, può accedere direttamente alle proprietà del Member (ed eredita anche il suo metodo *initialize()*). Questo semplifica molto il codice dell'Adapter. Semplifica anche il codice del client. Anziché creare un nodo e poi usarlo per creare l'Adapter, il client può semplicemente creare l'Adapter:

```
giangigi = MemberNode2.new("Giangigi", "Rossi", nil)
pieruga = MemberNode2.new("Pieruga", "Verdi",
                                            giangigi)
giorgiarmando = MemberNode2.new("Giorgiarmando",
                                   "Bianchi", giangigi)
piermalloppo = MemberNode2.new("Piermalloppo",
                                     "Gialli", pieruga)
filiberto = MemberNode2.new("Filiberto", "Magenti",
                                            pieruga)
```

Questa variante del pattern si chiama Class Adapter.
Nel nostro caso, un Class Adapter ci permette di risparmiare codice. A volte, però, un Object Adapter è più flessibile. Ad esempio un Object Adapter ci permette di "avvolgere" non solo un *Member*, ma anche un oggetto di una sottoclasse di *Member*. Se *Member* avesse molte sottoclassi, ci servirebbe un Object Adapter per avvolgerle tutte senza scrivere un Adapter specifico per ciascuna. Come sempre, non esistono ricette buone per tutte le circostanze. La scelta sta a voi. Quanto a Salvatore, anche per questa volta il suo fine settimana è salvo.

SE C'È IL COMPILATORE DI MEZZO

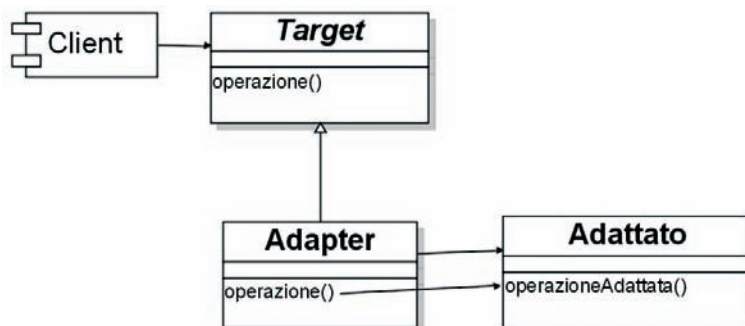
Il pattern Adapter è particolarmente adatto ai linguaggi staticamente tipati. I linguaggi dinamici come Ruby ci costringono a guardare nel codice per ricostruire le interfacce a cui "adattarci", in un linguaggio statico come Java le interfacce sono dichiarate esplicitamente. Una versione Java del TreePanel parlerebbe probabilmente ad un'interfaccia Node. Il nostro MemberNode dovrebbe quindi non solo includere un Member (se si tratta di un Object Adapter), ma anche implementare Node:

```
public class MemberNode implements
                                          Node...
```

Lo stesso vale per un Class Adapter: MemberNode2 dovrebbe ereditare da Member ed implementare Node:

```
public class MemberNode2 extends
            Member implements Node...
```

Ecco che aspetto ha un Class Adapter in un linguaggio staticamente tipato. Il Target è l'interfaccia alla quale il client vuole parlare, nel nostro caso Node:

Client
Target
operazione()
Adapter
operazione()
Adattato
operazioneAdattata()

CONCLUSIONI

Il pattern Adapter è utile in molte circostanze. Come abbiamo visto, è la soluzione ideale per collegare pezzi di codice scritti da autori diversi. Ma non solo: spesso ha senso usare un Adapter anche se abbiamo il controllo sia del client che dell'oggetto adattato, ad esempio per separare diversi pezzi della nostra architetture. Probabilmente avete già usato questo pattern per scopi simili in passato, anche se magari non gli avete dato un nome. Ovviamente questo tipo di "adattatore" fa in modo di poter riunire sotto un unico cappello software che in teoria non potrebbe comunicare. In questo modo ci assicuriamo che eventuali framework, spezzoni di codice, appunti, siano sempre integrabili e velocizziamo di molto il nostro lavoro. D'altra parte il principio base della programmazione è la riusabilità. Nei prossimi numeri affronteremo ancora argomenti legati ad un principio teorico come i pattern, che però hanno un risvolto decisamente pratico

Paolo Perrotta

SOFTWARE SUL CD

Virtual Box 1.3.8

IL VIRTUALIZZATORE FREE

La virtualizzazione è l'ultima frontiera. La nutrita schiera di sistemi operativi con le relative versioni, rende particolarmente complesso lo sviluppare programmi in grado di funzionare sui vari sistemi senza problemi. Per questo ogni programmatore tende a testare le proprie applicazioni su più sistemi. E' ovviamente quasi impossibile disporre di tante macchine per quanti sono i sistemi operativi così ci vengono in aiuto i software di virtualizzazione che emulano il comportamento di una macchina via software.

VirtualBox è uno di questi. È recente, è molto leggero ed è free.

virtualbox/VirtualBox_1.3.8_Win_x86.msi

APACHE 2.2.4

IL WEB SERVER PIÙ USATO AL MONDO

L'ultima versione del Web Server progettato da Apache e che da solo tiene in piedi una buona parte di Internet. Nonostante l'agguerrita concorrenza Apache rimane

un Web Server incredibilmente usato. I suoi moduli sono praticamente illimitati, è incredibilmente leggero,viene utilizzato praticamente su tutte le piattaforme di hosting al mondo. Se avete in mente di progettare un'applicazione Web non potete fare a meno di avere installato sulla vostra macchina in locale una versione di Apache. Quella che vi presentiamo è la versione 2.3 della serie 2, che ormai è sufficientemente consolidata da essere usata in ambienti di produzione.

Apache/apache_2.2.3-win32-x86-no_ssl

IRRLICHT 1.2

ACCENDI IL MIGLIORE MOTORE 3D OPEN SOURCE!

Irrlicht è un motore per la grafica tridimensionale, scritto in C++ e utilizzabile sia con questo linguaggio, sia con la tecnologia .NET.Presenta le principali caratteristiche di un motore professionale e vanta una notevole comunità di sviluppatori, con diversi progetti in attivo. Irrlicht ha tra i suoi pregi anche quello di potere utilizzare sia DirectX che OpenGL per cui diventa particolarmente adatto anche per lo sviluppo di giochi multipiattaforma

irrlicht/irrlicht-1.2.zip

CRYSTAL SPACE 1.0

IL RE C'È ANCORA

E' stato per lungo tempo il dominatore delle scene per quanto riguarda gli Engine 3D. Lo è stato almeno fino all'avvento di Irrlicht e del più recente XNA. Questo non vuol dire che non rappresenti ancora lo stato dell'arte nel campo della

programmazione dei videogiochi. Ad oggi si tratta di uno degli engine 3D che conta il maggior numero di installazioni e giochi sviluppati. In questo numero vi presentiamo la release 1.0 di cui purtroppo sono disponibili solo i sorgenti. Tuttavia la compilazione non è un'operazione complicata e vi mette in grado di conoscere ancora meglio il prodotto su cui state lavorando

crystalspace/crystalspace-src-1.0.1

DEV C++ 4.9.9.2

UN EDITOR C++ A BASSO COSTO

Dev C++ è un editor distibuito su licenza GPL, come tale non ha costi relativi al diritto d'autore. Come tutto o quasi il software GPL la sua economicità non è affatto sinonimo di scarsa qualità. Al contrario DEV C++ è uno degli editor C++ più amati ed utilizzati da chi sviluppa in C++. Le caratteristiche sono notevoli. Si va dal debugger integrato, al project management, al class browser, al code completion. L'insieme di queste caratteristiche unite all'eccezionale leggerezza dell'ambiente lo rende particolarmente comodo da utilizzare per sviluppare progetti C++ anche di grandi dimensioni

devcpp/devcpp-4.9.9.2_setup.exe

JAVA SE DEVELOPMENT KIT 6

IL COMPILATORE INDISPENSABILE PER PROGRAMMARE IN JAVA

Se avete intenzione di iniziare a programmare in Java oppure siete già dei programmatori esperti avete bisogno sicuramente del compilatore e delle librerie Java indispensabili. Sotto il nome di Java SE Development Kit vanno appunto tutti

gli strumenti e le librerie nonché le utility necessarie per programmare in JAVA. L'attuale versione è la 6.0, ovvero la nuovissima release densa di innovazioni e molto più legata al desktop di quanto non fossero tutte le precedenti

j2se/jdk-6-windows-i586.exe

ECLIPSE SDK 3.2.2

L'IDE TUTTOFARE

Eclipse è un progetto completo portato avanti da Eclipse Foundation con la collaborazione di una miriade di aziende fra cui IBM, Adobe, Sun e che si è prefissata lo scopo di creare un IDE estendibile per plugin adattabile a qualunque tipo di linguaggio o tecnologia. Di default Eclipse si propone come IDE per Java ed è qui che da il meglio di se. Ma proprio grazie ai suoi plugin è possibile utilizzarlo come ambiente di programmazione per PHP, per C++, per Flex e per molti altri linguaggi ancora. Inoltre sempre grazie per ciascun linguaggio sono disponibili altri plugin ad esempio per rendere l'ambiente RAD o per favorire lo sviluppo dei Web Services o altro. Insomma lo scopo è stato raggiunto completamente. Eclipse è realmente un IDE tuttofare, ormai maturo, e che serve una miriade di programmatori grazie alle sue caratteristiche di affidabilità e flessibilità. Unica nota negativa: una certa pesantezza che lo rende idoneo ad essere usato solo su PC con una dotazione hardware minima di tutto rispetto

eclipse/eclipse-SDK-3.2.1-win32.zip

PHP 5.2.1

IL LINGUAGGIO DI SCRIPTING PIÙ AMATO DEL WEB

Sono tre le colonne portanti di Internet: PHP, APACHE e MySQL. Certo la concorrenza è forte. Asp.NET e SQL Server avanzano con celerità, ma a tutt'oggi non si può affermare che i siti sviluppati in PHP costituiscano la stragrande maggioranza di Internet. Quali sono le ragioni del successo di cotanto linguaggio? Prima di tutto la completezza. PHP ha di base tutto quello che serve ad un buon programmatore, raramente è necessario ricorrere a librerie esterne, e quando è proprio indispensabile farlo esistono comunque una serie di repository che rendono tutto immediatamente disponibile ed in forma gratuita. Il secondo punto di forza del linguaggio sta nella sua capacità di poter essere utilizzato sia in modo procedurale che nella sua forma ad oggetti certamente più potente e completa. Esiste un terzo di punta di forza essenziale che è quello riguardante la curva di apprendimento.

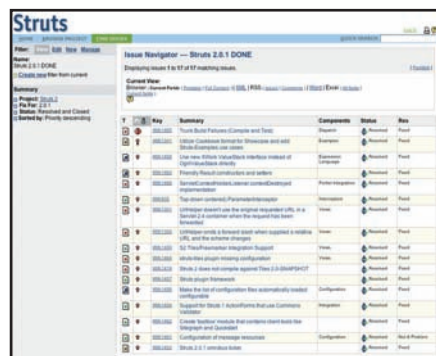
PHP è in assoluto uno dei linguaggi con la curva di apprendimento più bassa nel panorama degli strumenti di programmazione. Si tratta perciò di uno strumento indispensabile per chi si avvvicina alla programmazione web, a meno che non intendiate scegliere strade diverse quali possono essere ASP.NET o JSP

php/php-5.2.0-Win32.zip

STRUTS 2.0.1

IL FRAMEWORK MVC PER JAVA

Il pattern MVC è probabilmente il piùusato da tutti gli sviluppatori in ogni linguaggio.Questo framework è il leader per quanto riguarda lo sviluppo di applicazioni Java secondo il pattern MVC. Molto comodo, stabile e affidabile, rappresenta la base di partenza per applicazioni che devono essere facilmente manutenibili.Uno standard da usare se progettate applicazioni di dimensioni generose, ma anche quando si sviluppano web application professionali

struts/struts-2.0.1-all.zip

TOMCAT 6.0.9

IL SERVLET CONTAINER PER JAVA E JSP

L'idea è molto semplice. Sviluppare im Java pagine Web. Ad un primo sguardo, Tomcat potrebbe sembrare un normale Web Server. Ed in effetti è un normale Web Server! In grado di soddisfare le richieste per qualunque pagina HTML. In realtà però Tomcat è anche qualcosa in più, ovvero la capacità di soddisfare le richieste per applicazioni Java. Potrebbe sembrare complesso, in realtà lo è meno di quanto sembri. Immaginate Tomcat come un grande contenitore al cui interno ci sono altri contenitori ciascuno dei quali rappresenta un'applicazione Java, che una volta richiamata costruisce una

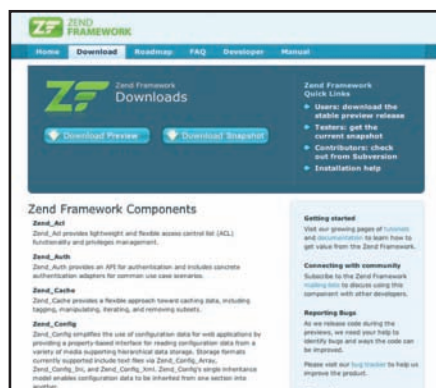
pagina Web interpretabile da un browser. Questo consente di sviluppare pagine Web utilizzando tutta la potenza della normale gerarchia delle classi Java e la sintassi e il linguaggio che qualunque programmatore Java conosce bene.

tomcat/apache-tomcat-6.0.9.exe

ZEND FRAMEWORK 0.9

L'SDK EVOLUTO PER PHP

Chi sviluppa in PHP è abituato a sviluppare in maniera autonoma il proprio framework, partendo dalle proprie esperienze e necessità. Questo ha però dato origine ad un ecosistema di applicazioni spesso difficilmente manutenibile. Zend ha sviluppato un proprio Framework completo che dispone di una serie di meccanismi che velocizzano la risoluzione dei problemi più frequenti. Si va dall'implementazione del pattern MVC alla creazione dei Web Services, alla gestione delle stampe in PDF. Si tratta di un framework piuttosto affidabile essendo sviluppato da Zend che è anche la software house promotrice dello sviluppo di PHP.

zendframework/ZendFramework-0.9.1-Beta

RUBY.NET BETA 5

IL NUOVO CHE AVANZA

Ruby, probabilmente lo conosce tutti. E' un linguaggio nato ormai da una decina d'anni inizialmente con la pretesa di essere uno strumento matematico dalle caratteristiche avanzate. Nel tempo si è evoluto e migliorato fino a raggiungere oggi una piena maturità che lo colloca fra i linguaggi più usati dell'ultimo anno. Proprio uno dei framework più noti sul web: Ruby On Rails ha vinto l'anno scorso il premio come miglior framework per lo sviluppo internet esistente, a dispetto di molti colosssi dell'informatica che operano in questo campo ed a testimonianza della grande maturità che questo linguaggio ha raggiunto. La sintassi è semplice, il linguaggio elegante, completo e particolarmete versatile, la curva di apprendimento molto bassa. Si tratta di uno strumento rapido che può coadiuvare lo sviluppo tradizionale e perché no? in molti casi sostituirlo. In questo numero di ioProgrammo ve lo presentiam nella sua versione per .NET, particolarmente ottimizzato per Windows.

Ruby.NET.beta0.5.bin.zip

PYTHON 2.5

L'EX GIOVANE RAMPANTE

Python è stato considerato per lungo tempo il nuovo che avanza.
Attualmente non lo si può più definire in questo modo, Python è ormai un linguaggio stabile e completo che trova applicazione inun gran numero di progetti. Se ne parla sempre di più in campo industriale come su Internet.

Soprattutto un gran numero di applicazioni anche in ambiente Windows girano ormai grazie a Python e presentano interfacce grafiche ottimamente strutturate. Ciò nonostante Python rimane un grande linguaggio di scripting adatto a gestire in modo completamente automatico buona parte di un sistema operativo sia esso Linux o Windows

python/python-2.5.msi

MYSQL 5.1.15

IL PRINCIPE DEI DATABASE

|Indispensabile per programmare webapplication in tecnologia PHP. Nonche non sia possibile utilizzare altridatabase, ma MySQL e PHP rappresentano veramente un binomioinscindibile. L'integrazione fra questo database e il linguaggio di scripting più usato sulla rete è talmente alta da fare divenire quasi un obbligo l'uso congiunto di questi due strumenti

Directory/mysql-5.0.27-win32.zip

FIREBIRD 2.0.0.12748-0

IL DATABASE VELOCE COME UN FULMINE

Ha attraversato una serie incredibile di disavventure. Acquistato da Borland è poi tornato ad essere OpenSource. Presente sul mercato da tempo immemorabile è riuscito a sopravvivere alle sue varie vicissitudine solo grazie alle sue grandi doti tecniche. E' un database velocissimo e ultraleggero, dotato però di tutte le funzioni di un server professional

Firebird/Firebird-2.0.0.12748-0-Win32.exe

POSTGRES 8.2.3

IL GRATIS COMPLETO E VELOCE

Nessun server di database offre la completezza delle funzioni esposte da PostgreSQL e rimane comunque completamente Gratis. PostgreSQL è probabilmente il più estendibile fra i database esistenti. Inutileparlare della gamma praticamente completa delle sue funzioni. Il punto di forza che lo pone probabilmente al di sopra di tutti i concorrenti rimane l'alta possibilità di personalizzazione, oltre, naturalmente, alla velocità, alla stabilità ed al costo nullo.Unica pecca, una certa complessità nella gestione. E' sicuramente da usare in ambienti di produzione professionali che non possono accontentarsi di alcun compromesso

postrgresql/postgresql-8.2.1-1.zip

AUTOMI CELLULARI: IL GIOCO DELLA VITA

TRA GLI AUTOMI CELLULARI, IL PIÙ NOTO È PROBABILMENTE IL GIOCO DELLA VITA. UN SEMPLICE E BEN STRUTTURATO ESEMPIO DI SISTEMA AUTO-ORGANIZZATO UTILE PER COMPRENDERE MOLTI FENOMENI ANCHE NATURALI

Per il programmatore il primo approccio con il gioco della vita o come è meglio conosciuto con "the game of life" o semplicemente "life" è spesso un esercizio per approfondire i dati strutturati omogenei bidimensionali. È un'infallibile strumento nelle mani del docente che riesce ad attrarre lo studente con un gioco per trasmette il fondamentale concetto di matrice. Il termine "gioco" non deve distogliere né sminuire la valenza di questo importante progetto, poiché grazie a life e ad alcune sue varianti è stato possibile studiare alcuni fenomeni naturali e sociali. Come vedremo, la molteplicità di configurazioni che si possono ottenere e che spesso producono risultati grafici sorprendenti ha stimolato lo spirito "artistico" di molti programmatori o semplicemente appassionati che si sono avvicinati a questo mondo. Insomma, un modo stimolante per trattare i sempre più diffusi automi cellulari. In questo articolo presenterò a partire dalle regole del gioco tutto ciò che serve per realizzare un programma per computer utile a svilupparlo. Inoltre, osserveremo del software libero già pronto per trattare life ed esploreremo, infine, alcune interessanti varianti del gioco.

DAGLI AUTOMI CELLULARI A LIFE

Life, anche se così definito, non è un gioco nel senso comune che attribuiamo alla parola. Sicuramente ha i suoi aspetti ludici e ricreativi ma presenta alcune peculiarità molto simili anche ad altri campi che non siano quelli del gioco. Si tratta di un automa cellulare. Brevemente, ricapitolando cosa è un automa cellulare, visto che è stato il protagonista dell'ultimo appuntamento di soluzioni. È un sistema ideato da uno degli studiosi più quotati del secolo nonché il padre del computer, mister John Von Neumann. Esso prevede un campo di celle generalmente disposte su uno spazio bidimensionale, in modo da definire una sorta di griglia. Le celle possono trovarsi in due stati, riconosciuti come vivo o morto (acceso o spento o ancora come zero o uno). Attraverso regole di riproduzione che tengono conto dello stato di ogni singola cella, rispetto a quello del suo intorno, viene esaminata l'evoluzione della popolazione nel tempo. Per questo si tratta di un sistema miniaturizzato che simula la vita artificiale. Ed anche se le regole di produzione sono poche e gli stati di ogni singola cella sono soltanto due, è stato dimostrato come sia complesso studiare l'evoluzione della popolazione, e come si possano innescare elementi di riproduzione inaspettati.

Le regole di evoluzione, insieme al numero di stati di una cella e alle caratteristiche del campo definiscono diversi tipi automi cellulari. Quello che intendiamo sottoporre alla nostra lente di ingrandimento è stato proposto da uno dei più estrosi matematici di fine secolo che più volte abbiamo incrociato nelle nostre analisi, si tratta di John Horton Conway. È stato lui ad introdurre "the game of life". Ma il risalto internazionale e la divulgazione alla platea scientifica e dei programmazioni è dovuta ad un'altra nostra conoscenza Martin Gardener che presentò gli studi di Conway in un memorabile articolo dell'ottobre del 1970 nella prestigiosa rivista Scientific American (si veda box "Life su web"). Come lo stesso Gardener qualche anno dopo ricorderà, il suo articolo rese popolare life ma fu anche un essenziale propulsore per la divulgazione degli automi cellulari fino a quel momento sottovalutati. Sebbene la presentazione di Conway all'epoca era più associabile ad un solitario che alla simulazione di un sistema evolutivo artificiale, presto le elaborazioni al computer avrebbero aperto il viatico verso sempre più numerose e raffinate applicazioni basate su life. Infatti, vale la pena di ricordare che la prima formulazione era stata fatta con carta e penna ed i rari elaboratori dell'epoca non venivano utilizzati per quelle che ancora sembravano applicazioni poco degne di automazione computerizzata.

LE REGOLE

È un gioco atipico poiché non sono presenti giocatori né tantomeno ci sono vincitori o perdenti. Si tratta di vedere come una configurazione iniziale (pattern)

di organismi sistemati su una griglia evolva nel tempo. Ed è proprio questo che diverte, la pratica di stabilire dei pattern che diano vita a popolazioni che hanno qualche motivo di interesse nella loro evoluzione temporale. Gli organismi, o usando il lessico più naturale degli automi cellulari: cellule o celle, possono trovarsi soltanto i due stati, vivo o morto. Lo stato di vita viene evidenziato e marcato sulla cella. La griglia è bidimensionale ed è fatta di tante celle come una scacchiera che però non ha, almeno teoricamente, limiti. Ogni cella ha un intorno di altre otto celle (intorno di Moore). Le regole di riproduzione che descrivono l'evoluzione sono state pensate da Conway per evitare questo processo in due situazioni estreme ma opposte ritenute negative e non naturali: il sovraffollamento e la solitudine. Non che le regole proposte siano del tutto naturali come vedremo! Le leggi di evoluzione della popolazione sono:

1. Una cella viva che ha due o tre vicini vivi alla generazione successiva risulterà ancora viva.
2. Una cella non in vita (o morta) nascerà alla generazione successiva se ha esattamente tre celle adiacenti vive;
3. Una cella viva che non si trova nella situazione 1 o 2, ossia non ha esattamente due o tre vicini vivi alla prossima generazione morirà. In particolare:

a. Una cella viva che non ha alcun vicino o che ne ha al più uno nella prossima generazione morirà per solitudine.
b. Una cella viva che abbia quattro o più vicini nella prossima generazione morirà per sovraffollamento.

Tutte le nascite e le morti prendono posto esattamente nello stesso istante, cosicché le celle morenti non possono partecipare alla nascita di altre celle (nello stato di morte). Così le celle che stanno nascendo non possono partecipare, nello stato di vive alla morte di altre celle. È questo l'aspetto di sistema discreto che esprime Life. Si tratta cioè di un sistema che evolve a "scatti" in fissati istanti di tempo, ad esempio ogni secondo. In **figura 1** è riportato un semplice esempio che riporta l'evoluzione di una popolazione inizialmente costituita da quattro celle per quattro generazioni.

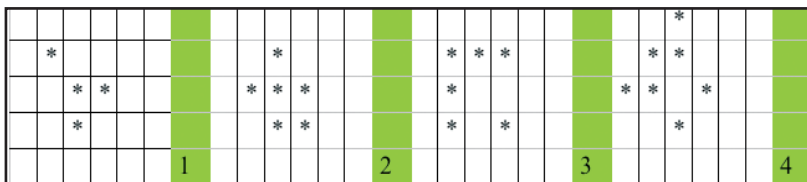
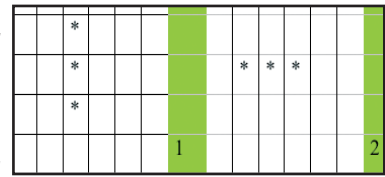



Fig. 1: "L'evoluzione di un pattern iniziale (1) per quattro diverse generazioni. La generazione (2) vede sopravvivere tutte le celle tranne la più esposta in alto a sinistra che muore per solitudine, inoltre in basso a destra si registra una nascita. La generazione tre (3) a partire dalla due (2) vede le celle centrali perire per sovraffollamento mentre si verificano nuove nascite. E così si prosegue.

MAX E I SUOI AMICI

Lo stato dell'arte e così avanzato a riguardo che si sono classificati alcuni schemi sia per avviare il processo (le configurazioni iniziali di celle vive) che terminali forme a cui si riducono le configurazioni; al fine di comprendere più facilmente come possa avvenire l'evoluzione. Vi sono ad esempio delle configurazioni che oscillano continuamente come il caso della **figura 2**.

Fig. 2: "Esempio di oscillazione continua tra le due configurazioni proposte"

È stato Weisstein che ha classificato tutti i possibili modi in cui si può ridurre una in modo stabile popolazione dopo il naturale processo evolutivo, ossia in modo che rimanga tale se non influenzata da altre celle che nell'intorno mutano. Il principale esempio è il blocco, quattro celle vive disposte in quadrato **figura 3**. Come si può notare sopravvivono tutte ma non danno origine a riproduzione.

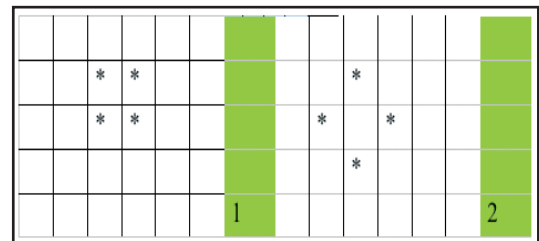



Fig. 3: "Schemi terminali stabili (1) block o blocco e (2) tube o tubo

Tutte gli altri schemi di Weisstein sono riportati in **figura 4**. Sono stati estratti da mathworld.wolfram. Colui o ciò che vedete in **figura 5** è invece Max, un misto tra uno spaventa passeri e un terminator, è un importante schema da cui partire per ottenere un tasso di crescita della popolazione quadratico. Dopo la classificazione degli schemi terminali si passa a quella degli schemi iniziali o pattern che producono evo-

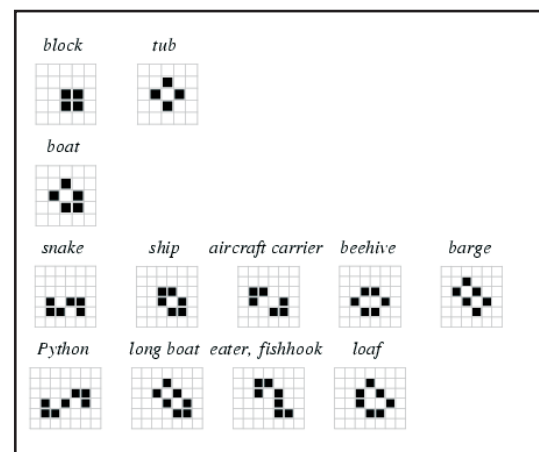



Fig. 4: "Insieme completo degli schemi terminali stabili proposti da Weisstein"

luzioni degne di essere osservate. Ad esempio, Max ha registrato l'importante risultato di crescere in modo quadratico, cosa che inizialmente si pensava impossibile. Il primo ad ottenere tale risultato fu Gosper con "breeder", che lavorava lasciando dietro di sé una coda di gun. Da allora sono state prodotti complessi schemi, tra cui porte logiche per glider, un sommatore, un generatore di numeri primi, e una cella che emula lo stesso Game of Life scalato nello spazio e nel tempo. Molti di questi schemi sono presentati nel software descritto nel paragrafo successivo.

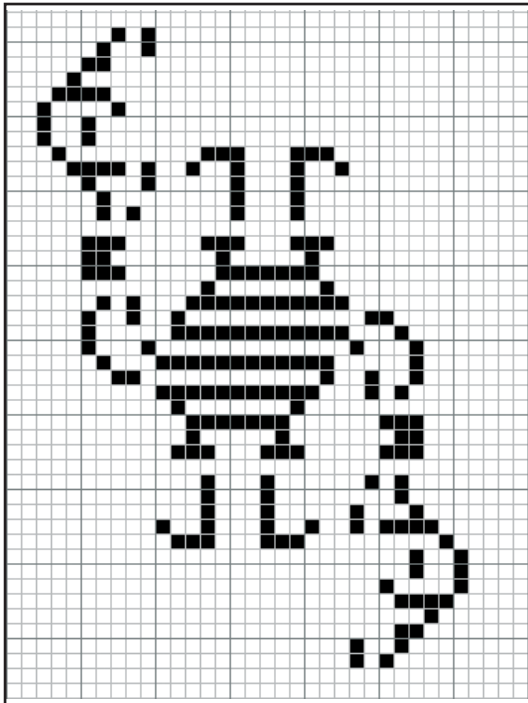



Fig. 5: "Il pattern Max che descrive un tasso di crescita quadratico"

SOFTWARE E IMPLEMENTAZIONE

Sviluppare un programma che realizzi il gioco della vita è un compito che può essere svolto a diversi livelli. Si parte come detto con un esercizio che il professore dà per far comprendere le matrici e si arriva a sofisticate applicazioni che gestiscono tutte le possibili regole e varianti del gioco e curino al meglio la rappresentazione grafica. Parliamo prima della seconda categoria. Si tratta di software sviluppati un po' per passione e un po' per scopi professionali che realizzano in molti aspetti ed efficientemente il gioco della vita. Molti esempi si trovano sul web dove sono riportate delle applicazioni come applet java che simulano al meglio il gioco. Tra queste (come segnalate nel box life su web) divertenti sono i life colorati. In giro per il web si trova anche molto software free che fornisce ben fatte applicazioni stand alone. La più nota, che troverete anche su CD, è life 32 che permette di produrre facilmente dei pattern e di provarli sia a passi che in modo fluido. In questa seconda modalità è possibile stabilire il tempo di transizione tra una generazione e l'altra. In **figura 6** è riportato una cattura dell'applicazione in esecuzione.

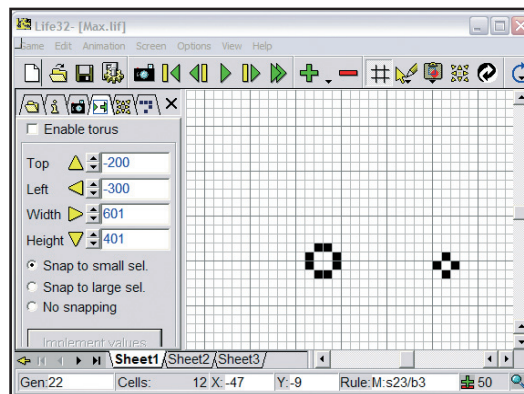



Fig. 6: "Life 32 in esecuzione"

Life32 permette di salvare e ovviamente caricare dei pattern in modo da lavorare anche in diversi momenti su uno stesso progetto. Si trovano infatti, sempre su internet collezioni di pattern, sul Cd ne è stata caricata una completa. Infine, ma non per ultima, infatti il software descritto e da provare e scoprire tante sono le funzioni e parametri che è possibile settare, segnalo la possibilità di cambiare le regole andando a specificare in modo semplice le richieste di sopravvivenza e di nascita, in termini di celle adiacenti (**Figura 7**) Come si può vedere si possono produrre sistemi che innescano comportamenti estremamente caotici come sistemi che registrano evoluzioni che si consumano nel giro di pochissime generazioni. Nel paragrafo successivo vi è un cenno delle varianti che si possono produrre.

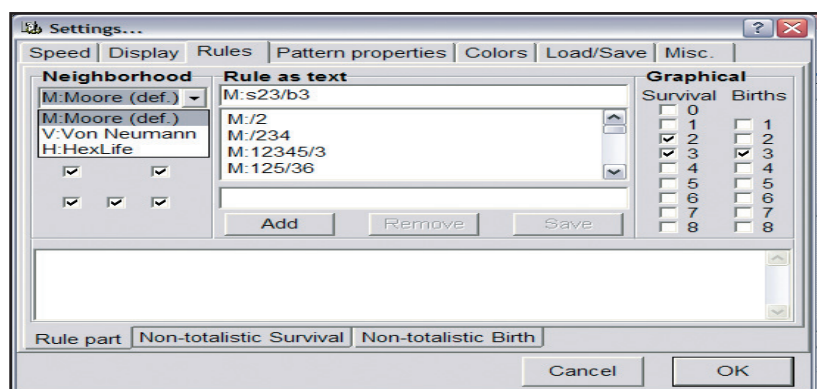



Fig. 7: "Life 32 in configurazione delle regole (file>rules)"

Se invece di essere utilizzatori vogliamo più opportunamente essere programmatori, dobbiamo realizzare un progetto che preveda un'ade-

guata struttura dati e una serie di routine in grado di comandare le richieste essenziali del gioco. La struttura che meglio si presta ad essere usata è certamente una matrice che sarebbe sufficiente se di booleani. Ma per prevedere varianti e life colorati e meglio predisporla di numeri interi; sia mappa tale variabile. Ogni cella, per la versione originale, può valere ad esempio 0 se morta e 1 se viva. Una programma a livello didattico può essere così strutturato:

```
...{
// inizializzazione e configurazione della mappa
configura(mappa);
visualizza(mappa);
do
    inizializza(nuovamappa);
    for (i=1; i<m; i++)
        for (j=1; j<n; j++)
            switch (contavicini(mappa,i,j))
            {case 0,1,4,5,6,7,8 : nuovamappa[i,j]<-false;
                                                    // morte
            case 2: nuovamappa[i,j]<-mappa[i,j];
            case 3: nuovamappa[i,j]<-true  // vita
            };
    copia(nuovamappa,mappa);
    visualizza(mappa);
    cout("Vuoi visionare altre generazioni (S/N)");
    leggi(risp);
while (risp='N');
}...
```

Ovviamente, si tratta di una versione basic in cui tutte le operazioni di configurazione e inizializzazione della mappa avvengono in sequenza. Inoltre è poco personalizzabile se non manipolando il codice. Nell'implementare le routine si possono introdurre elementi di grafica come per la visualizzazione della mappa che in questo caso risulta sicuramente più gradevole. Interessante a scopo didattico è contavicini sviluppata per l'intorno di moore

```
Int  contavicini(matrice mat; int riga,colonna);
{  int i,j;

   // Inizializza il contatore
   int Conta=0;
   // Esplora tutti i vicini
   for ( i=riga-1;i<riga+1; i++)
      for ( j= colonna-1; j<colonna+1; j++)
         if ((i>0) and (j>0) and (i<m+1) and (j<n+1))
            if (mat[i,j]==1) then conta++
   return conta
}
```

Andrebbe ottimizzato il funzionamento a bordo mappa.

IL MONDO DELLE VARIANTI

Nel corso degli anni sono state introdotte varianti all'idea originaria. Nella versione originale, un organismo nasce se ha esattamente tre organismi vivi vicini, sopravvive se ne ha 2 o 3, e muore altrimenti, ciò è simbolizzato con "s23/b3". Il primo numero, o lista di numeri, dopo la lettera s indica le richieste per la sopravvivenza (survival). Il secondo numero o lista di numeri, dopo la lettera b rappresenta le regole per la nascita (b come birth). Ad esempio "s25/b4" significa "un organismo nasce se ha 4 vicini vivi, e sopravvive se ne ha 2 o 5, muore altrimenti. HighLife è 23/36: in aggiunta alle regole originali 6 vicini comportano una nascita. HighLife è particolarmente conosciuto per i suoi replicanti. Su una discussione dell'enciclopedia libera sul web Wikipedia sono state catalogate con cura una numerosa serie di altre possibilità. Anche se, come sottolinea l'autore la maggioranza di questi universi alternativi è troppo caotica o troppo desolata. Eccoli:

- s/b3 (stabile) quasi ogni cosa è una scintilla
- s5678/b35678 (caotico) diamanti, catastrofi
- s/b2 (esplosivo) "Seeds" phoenix, minimale
- s/b234 (esplosivo) phoenix, lacey patterns
- s12345/b3 (esplosivo) schemi labirintici
- s125/b36 (caotico) blocchi 2x2 simili all'originale
- s1357/b1357 (esplosivo) tutto è un replicante
- s1358/b357 (caotico) un'ameba bilanciata
- s23/b3 (caotico) "Conway's Life"
- s23/b36 (caotico) "HighLife" (ha replicanti)
- s235678/b3678 (stabile) macchie d'inchiostro che si asciuga subito
- s235678/b378 (esplosivo) coagulazioni nel caos
- s238/b357 (caotico) broken life

Fonte per questa lista di regole: Life32

Altre variazioni si basano sulla modifica di altri elementi del gioco, come la dimensionalità del campo, che può essere a una come a tre dimensioni, o lo stato dei singoli organismi che può appartenere a più di due stati. In questo ultimo caso si da origine a Life colorati (un colore diverso per ogni stato) che producono evoluzioni temporali spesso molto affascinanti anche da un punto di vista grafico. Oltre alla dimensionalità anche la forma della cella può dare origine a nuove varianti, ed ecco che ad esempio rispunta la struttura di Hex un gioco di strategia esaminato in passato tra queste pagine in cui la singola cella è un esagono. Qui il concetto di adiacenza è più naturale visto che ogni cella è esattamente confinate con sei altre sei, con ognuno dei sui lati; non come l'intorno di moore per celle quadrate dove alcune adiacenze si hanno per un solo punto.

Fabio Grimaldi